AN ARABIC PROGRAMMING ENVIRONMENT

By

ABDULMALIK S. AL-SALMAN

Bachelor of Science
King Saud University
Riyadh, Saudi Arabia
1988

Master of Science
University of Georgia
Athens, Georgia
1992

Submitted to the Faculty of the
Graduate College of the
Oklahoma State University
in partial fulfillment of
the requirements for
the Degree of
DOCTOR OF PHILOSOPHY
July, 1996

AN ARABIC PROGRAMMING ENVIRONMENT

Thesis Approved:

Thesis Advisor

H. Lu

Thomas C. Collins

Dean of the Graduate College

ACKNOWLEDGMENTS

Praises be to ALLAH, the Lord of the universe, without his bounty I would not have come to Oklahoma or finished this work.

I wish to express my deep gratitude and sincere appreciation to my major professor, Dr. K. M. George, for his scientific guidance, moral support and continuos encouragement and understanding. My thanks are extended to the members of my advisory committee, Dr. G. E. Hedrick, Dr. K. Gasem, and Dr. H. Lu, for their valuable suggestions and encouragement during the course of this project.

I would also like to give a special appreciation to my parents and my wife for their support and encouragement all the time. I am also thankful to all my friends in Stillwater who made my staying here very enjoyable.

I also forward my thanks to my home country, Saudi Arabia, the land of goodness.

TABLE OF CONTENTS

Appendix **Page**

LIST OF TABLES

LIST OF FIGURES

NOMENCLATURE

APC	Arabic Pascal Compiler
APE	Arabic Programming Environment

CHAPTER 1

INTRODUCTION

Throughout the world, computers play an important role in people's life. The concept of literacy starts to shift from not knowing how to read and write to not knowing how to operate and utilize computers. For instance, in Saudi Arabia many houses have computers either to be used for scientific purposes or just for entertainment. Computer classes have been introduced in all high schools. Education planners in Saudi Arabia are nowadays thinking seriously of introducing computer classes not only in middle schools, but also in elementary schools [Min87, DAM92, Gne94, FE95, GFA95, Man95]. The main difficulty facing such planing is the lack of good skills in English language prior to college. Thus, to enhance computer teaching in Arab countries is to Arabize computers in all levels: hardware interfaces such as operating systems, software interfaces such as applications, and programming languages.

Every Arab national feels and appreciates the extreme importance of computer Arabization. Computer Arabization can be defined as the conversion of well-tested and well-proven computer software to enable Arabic-speaking users to utilize it effectively and efficiently. This work is a small contribution to this end.

This dissertation is comprised of six chapters, including this introduction. The rest of this chapter states the main objectives and motivations. It also defines the design guidelines for developing the Arabic Programming Language Environment. Chapter 2 is devoted to explain the main characteristics of the Arabic language. Understanding these characteristics is essential for designing an Arabic software. Chapter 3 discusses related work. Chapter 4 describes the design and implementation of the Arabic Programming Environment (APE). Chapter 5 explains the design and implementation of the Arabic Pascal Compiler (APC). The APE and APC complement each other. Chapter 6 concludes the work and states the avenues for future work.

The dissertation also has five appendixes. Appendix A lists and explains the APE messages. Appendix B lists the words and symbols that are used in the Arabic Pascal language. Appendix C shows the syntax diagrams for the Arabic Pascal language. Appendix D lists and explains the compilation error messages and the run-time error messages. Finally, Appendix E lists a set of test programs used as a validation suit.

1.1 OBJECTIVES AND MOTIVATION

The main objective towards the Arabization of computing is to be able to write programs in Arabic. Arabic programming languages and their environments should be available as other international programming languages to enhance the advancements in the Arab world. Programming language environment is an integrated set of tools for program development such as editor, compiler, linker, and debugger. So, it is substantially different from any Arabic word processors available in the market. Arabic language, in general, has many unique features that makes it different from all European languages. Consequently, developing an Arabic environment is different from developing an environment for other languages. Because of the importance of understanding the Arabic characteristics, we have dedicated a chapter, Chapter two, for describing those characteristics.

Arabic Pascal design and implementation was motivated by the need of a simple, structured, and powerful Arabic programming language to be used as a general purpose programming language, especially in teaching programming concepts. From an educational point of view, it is obvious that learning new concepts is more effective if the native language is used. Few years ago, such a project was not applicable because there were no mature Arabic environment systems that can handle efficiently the Arabic characters. Moreover, there was no need for such project because most computer users and programmers were educated adults who knew the English language and could easily program in English. However, this situation has changed.

1.2 DESIGN GUIDELINES

Several key factors influence the design of the environment. The Arabic environment should have the ability to read and display Arabic characters. Moreover, everything in the editor (writing direction, menus, tool-bar, status-bar, etc.) should be presented from right to left to accommodate the Arab users. These two features pose the major obstacle to developing such an environment. Also, to address a wide group of users and programmers, this work has to be built on top of a popular machine that can support Arabic character set. This led to using IBM PC and compatibles which are popular and widely available machines with numerous clones. They also support a reasonable Arabic interface under both MS-DOS and MS-Windows.

Since MS-Windows has a better user interface than MS-DOS, MS-Windows is chosen as the base platform for the work. MS-Windows version 3.0 and later can support Arabic characters with some applications for Arab users. What remains to be done is to develop some tools that are able to handle right alignments. The Arabic programming environment can be viewed as an editor complemented with compiling and running facilities. The editor should accept Arabic text with basic tools for editing, such as opening a file, saving it, cutting, copying, and pasting. Chapter 4 explains in detail the APE interface and its capabilities.

To address wide group of programmers, The selected programming language has to be well-known in its original design. Therefore, the shift from English programming language environment to Arabic programming language environment has to be minimal at least in this stage of Arabization. May be later, when many programmers get a long with Arabic programming languages, more independent Arabic programming languages can be designed. We have chosen Pascal language as a model to follow for many reasons. Among these are:

- Pascal is the most widespread programming language in the education field in Saudi Arabia and to some extent in other countries as well.
- Pascal is a fairly small language with relatively simple syntax that can easily be understood.
- Pascal is a structured language.

Instead of defining a new programming language that may have no link to any existing language, we chose to follow the standard Pascal [Coo83, Tib81]. By doing this, the ability to switch from English Pascal to Arabic Pascal becomes easy for teachers who know English Pascal and want to teach their students the Arabic Pascal. Similarly, switching from Arabic Pascal to English Pascal becomes easy for students who want to pursue the field of programming language or want to study some other English programming languages like C or FORTRAN. The Arabic Pascal, however, is not oriented to any specific purpose or group; it is a general purpose programming language that can be utilized by beginners as well as experts.

Translating the English Pascal to Arabic Pascal can be done in an easy, but inefficient way, that is translating the Arabic syntax (source code) to English and then feeding it to the regular English Pascal compiler. Unfortunately, this approach does not solve the problem of English difficulties due to the following reasons:

- Users will receive all compiler messages in English. These are difficult to be understood by a person who is not proficient in English.
- Debugging the source program is almost impossible.
- There is no way to improve or modify the programming language to suit the Arabic native speakers.
- It is important to build a complete and coherent project that contains only Arabic interface.

Accordingly, the Arabic Pascal language and compiler should be a stand alone software, not a front end to any Latin high level programming language.

CHAPTER 2

CHARACTERISTICS OF THE ARABIC LANGUAGE

To effectively build a good Arabic software, one should fully understand the main characteristics of the Arabic language. The Arabic language has many unique features that are different from many other languages. The following is a list of the main characteristics of the Arabic language [AA89, AA93, Ben87, Cas85, EEN89, Ham89, Ham94a, KH93].

1. Arabic, like other Semitic scripts, is written and read from right to left.
2. While text is written from right to left, numbers can be written and read from right to left or from left to right. Most Arabs read numbers and write them from left to right.
3. It has 28 basic characters.
4. In addition to the basic 28 characters, Arabic has special letters such as

   آ،لا،ى،ة

   and HAMZA letters such as

   ء، أ، ؤ، ئ، لأ

5. Even though the basic alphabet consists of 28 letters, it is extended to some 90 elements by additional shapes, marks, and vowels [TA90].
6. Some Arabic characters can be fused to form new shapes. For example, 'لآ' is composed of 'ل' and 'آ'. 'آ' is also composed of 'أ' and 'أْ' [TA90].

7. Arabic is a cursive script --like English handwriting-- links the letters of a single word.
8. Arabic script is extremely complex due to the context sensitivity of its written characters. Letters have up to four different forms: isolated, at the beginning, in the middle, and at the end of a word.
9. Most of the characters have three shapes depending on their position within the word.
10. All characters are consonants except the letters Alif, Waw, and Ya [Sim90]. Actually, these three characters are consonants with attached vowel sounds.
11. Most characters (17 out of 28) have a dot, two dots, or three dots associated with the character and can be above, below, or inside the character.
12. Arabic also may use stretch character forms to add emphasis or aesthetics to the written word.
13. No upper or lower cases exist in Arabic.
14. Arabic has vowels. Vowels are special shapes with Arabic words. It is good to have them in writing to avoid possible ambiguity, but they are not necessary in most cases. The vowels can be inferred from the context. Nowadays, most writers ignore them [Bec87].
15. Similar to accent marks in European languages, Arabic has diacritical signs. These signs are used to mark short vowels and emphasize or loosen a letter's sound [TA90].
16. A special feature of the Arabic alphabet is the large number of diacritics, points, and slashes, which are used to differentiate consonants or to represent vowels.
17. Some groups of characters have the same shape with slight change in the number of dots or the position of these dots [Jam91]. The dots and their positions are important in distinguishing between different characters [Gor90].
18. The Arabic characters are written cursively with frequent vertical overlap in handwritten form. For example, the first three characters of 'محمد' can be written by

hand above each other. Even in the printed characters, vertical overlapping is clear in the first two characters in the last example.

19. The pronunciation of the character, and hence the meaning of the word containing that character, differs according to the diacritic imposed over it. For example, 'جُمَـل' (sentences) is completely deferent from 'جَمَـل' (camel) even though they have the same set of characters.

20. Arabic, like Latin, is a synthetic language rather than a predominantly analytical language, such as English [FF93]. That means that the syntactical relationship of nouns, for instance, is indicated by case endings and that verbs are inflected by means of prefixes, infixes and suffixes to indicate the various persons, numbers, genders, derived forms, moods and tenses. On the other hand, a language such as English a separate noun or pronoun is required to indicate the person, number, etc.

21. A major characteristic feature of the Arabic language is that most of its words are built up from roots, whereas English words are generated from stems [AA94]. Surface affix removal processes produce word stems while deep affix removal processes produce word root [Alk91].

22. The key to searching the Arabic lexicon is the root of the word, whereas the stem is the key in English. A lexicon is a list of words in which each word is associated with its syntactic properties [AA94]. The most important property is the syntactic category such as whether the word is a verb or a noun.

23. The Arabic language is an inflectional language whereas English is an analytic language [AA94]. The derivation in Arabic is based on morphological patterns and the verb in Arabic plays a greater inflectional role than in English.

24. Particles and affixed pronouns can be combined to Arabic words. This characteristic is not available in English.

25. The structure of Arabic language is different from that of Indo-European languages, including English. For instance, the phrase "PACKED ARRAY" should be reversed in Arabic to "مصفوفة مربوطة". This is because in the Arabic language the adjective follows the nominal phrase, contrary to the English.

26. The Arabic language accepts the principle of abbreviation. Both types of abbreviation, forming a word from the first letters of a sequence of words and selecting a prefix or a postfix from a word was popular in many Arabic works of literature. For examples, Arabs used to write "صلعم" to stand for "صلى الله عليه وسلم", "اهـ" as a an abbreviation for "انتهى", and "ثنا" as a postfix for "حدثنا".

27. Finally, Arabic is highly flexible in word order, whereas English imposes a large number of constraints on word order.

Not all of the above characteristics affect the Arabic Pascal language and its environment. The features that deal with the formation of words, the structure of sentences, or the different shapes of characters are not directly related to this work. However, the following issues are affected by the characteristics of the Arabic language.

- The Arabic Pascal environment writes text from right to left. For numbers, we have adopted the MS-Windows approach. This approach lets the cursor move to the left while the newly entered digit appears to the right of the previous digits. In other words, the user starts by typing the most significant digit. This approach may cause some editing problems (deleting, copying, selecting, etc.) because the cursor position does not match with the insertion position. However, this approach simulates the way of writing by the majority of Arabs.
- For the sake of simplicity and flexibility, the Arabic Pascal does not accept the diacritical signs. Most Arabs easily can read a text without the need for diacritical signs.

- Even though the standard Pascal does not allow the underscore character as part of identifiers as well as keywords, the Arabic Pascal does. The reason is that the Arabic is a script language (links the letter of a single word). Unlike English, it is hard to read the concatenation of two words without a space or an underscore. For example, the translation of "GOTO" can be either "اذهبإلى" or "اذهب_إلى". The former translation is hard to read. The underscore character is inserted between concatenated words to increase the readability.
- The Arabic Pascal applies the principle of abbreviation whenever applicable. For example, the translation of "EOF" is "نها_م" which stands for "نهاية_ملف". Another example is "جتا" as a translation for "COS".
- The letter "ا" is sometimes used instead of "أ" or "إ". To give some kind of flexibility to the Arabic Pascal, the letter "ا" can be used instead of "أ" or "إ" for keywords only, not user identifiers. The same criterion is applied for "ة" and "ه" at the end of keyword, and "ي" and "ى" also at the end of keyword* . Therefore, "إذا" and "اذا", "مجموعة" and "مجموعه", "حرفي" and "حرفى", etc. are the same keywords.

* In fact, the letters are not interchangeable from the orthagraphic point of view, but some Arabs use them as interchangeable letters. The Arabic Pascal tolerates this common mistake.

CHAPTER 3

RELATED WORK

Software developed for English and other European languages may not be convenient for Arab users because of the nature of the language and its writing system. For example, Arabic editing environments are different from European ones. These differences exist not only in character shapes and direction of writing, but also in language structure as mentioned in Chapter 2.

The development in computer Arabization may also serve many other languages that are derived from the Arabic language or at least use a similar character set, such as, Farsi, Urdu, Malay. Therefore, almost all applications that are developed for the Arabic language can be adapted easily to these languages.

Arabized computers become essential in the Arab world with the widespread use of computers in education, business, and other aspects of life. However, the use of computers in many Arab countries was, and may still be, in its infancy compared with the West. A mission report published by the United Nations Development Program Regional Bureau for Arab States in 1984 stated that all areas of computer applications are missing except to some degree in management business applications [UN84]. This mission described that there is a very little utilization of computers in health, education, biology, agriculture, physics, geology, mining, construction, project management, and engineering design.

Mahjoub and Mandorah [MM87] explained the main reasons for the slow development of computer sciences in the Arab world as:

1. weak educational programs in the universities,
2. absence of national programs to introduce technology to the public,
3. lack of good Arabic software,
4. lack of Arabic text books and references,
5. high cost of Arabic computer systems, and
6. weak research activities in computer science with lack of communication and coordination among the Arab researchers.

Arabization is the process of making computers accessible to the Arab users by utilizing only the Arabic language. Since the Arabic language has several unique characteristics, which differ from the English or Latin languages, the Arabization of computers is significantly different from English or Latin modes. There are many aspects or avenues where the Arabization of computers took place. Among these aspects are standardization [MM87, Smi90, Bak87], information systems [Mou87, BKN86, AG90, Mor90, Tek90, Ali90,Alk91, Abu92, Alk92, Ham94b], pattern recognition [Ami88, ES89, Gor90, AU92, Aly89, Jam91, AHD94, Mah94, AMG94, EG88, GUA92, NA94, AFC94, AG95], artificial intelligence and expert systems [NEE88, EEN89, AH88, Ibr91, Als92, AA93, KH93, ELS93, QHA93, MM94], Arabic morphological analysis [Cha87, EH89, RS93, FF90, Fed92, FF93, AMA94a, AA91, AA94, Ali87, Ali88, Ali89, Ali90, NH94], input and output facilities [Cas85, Kha86, Del86, TNS86, Bec87, Smi90, Alk90, KC91], word processing (including spelling and grammar checking) [AA88b, Mic93, ELS93, Ham94a], operating systems [Kal85, MM87, Tay87, AA88a, Sau86, Ela90, Tab90, Mic92, Tab93], transliteration and translation [IDF89, AFC94, NM91, MM95, MT95], Islamic sciences [AM95, Ali87, AK88, MA94, Abd90, Ahm94, Hil90, Ala90, Alm90, Man90, MA94, Far90, Ald94], programming languages [MM87, Alkh, Najla, Bas95, AYE90, DM86, Ela86, CS92, CS95], and other fields in computer science [Del85, ZRN87, Ale94, EB90, AM94, Gne94, MA85, HE86, Bak86, Ela87, Far89, AM87, Mac90, Eld87, FE95]. The following three sections briefly discuss the status and the advancements in the most relevant issues to my work, namely: standardization, input and output facilities, and programming languages. The fourth section reviews the history of Pascal compilers. Before discussing these issues, we should point out that the accessibility of information and literature in the Arabization subject is difficult due to the lack of information exchange among the Arab countries and the lack of electronic (automated) information availability. For example, if a conference proceeding or a paper is published in one of the Arab countries, it is difficult to know about it or to access it unless it is distributed to other countries (which is not the case for many reasons, such as political situations).

3.1 STANDARDIZATION

There are at least three aspects in the Arabized computers that need standardization: character coding, keyboard layout, and Arabic computer terms. Unfortunately, there is not enough progress in the process of standardization in both character coding and computer terms. The absence of an agreed upon set of standards for character coding has a very negative impact on the process of Arabization. The keyboard has, to some extent, a standard layout inherited from the Arabic typewriters. The differences in the keyboard layout exist on the auxiliary marks and the special characters.

The absence of a responsible organization in the Arab world to take care of standardization led to many Arabic character coding systems. The first attempt to standardize the Arabic characters coding was done by Saudi Arabian Standards

Organization (SASO) in 1980 by calling for an international symposium [MM87]. However, the weak interaction between the researchers and manufacturers resulted in the hasty adoption of standards. The various features of the Arabic language coupled with the absence of an agreed set of standards for character coding and keyboard layout have led to several approaches in Arabization, some examples of these approaches can be found in [ASM82, Mah85, Man85, Dav86, Sau86, Mic88, Almua].

As yet, there is no standard code (ASCII) for Arabic characters. For example, NCR has almost six different character maps for its Arabic terminals and printers; and IBM has at least 12, seven of them at least for the IBM PC [*cf.* Smi90]. Arabized MS Windows has different character set from the Arabized MS DOS. In other words, an Arabic text that is written using 'EDIT' under MS DOS cannot be viewed with 'NOTEPAD' under MS Windows. This led to inconsistency in every software that needs to support both Arabic DOS and Arabic Windows.

Since the shape of the character is determined by the character itself and adjacent characters, there are two approaches for character coding:

1. Each character has a single code regardless of its shape.
2. Each character shape has its own code. In this case a single character may have up to four different codes. The total number of codes is over ninety.

Most computer systems use the first approach to avoid the large number of codings. The appropriate shape can be determined by a software depending on the contextual analysis.

On the other hand, the standardization of computer terms in Arabic language is not better than the standardization of character coding. An attempt to put some rules and boundaries for the Arabization of terms was conducted by Bakry [Bak87]. The failure in Arabized computer terms start even in the translation of the word "computer" itself. There are at least three translations: "حاسب", "حاسوب", and "كمبيوتر". The last translation is actually writing the word “computer” as is in Arabic (transliteration).The same criterion is applied for many other terms.

3.2 INPUT AND OUTPUT FACILITIES

An Arabic text must confront three main classes of problems when displayed on screen or printed by a printer [Del86, Bec87, Smi90]. The first class is caused by the Arabic cursive nature. The computer must be able to handle a sequence of isolated typed characters and display or print them in a smooth flow Arabic text. This process is called contextual formatting. Automatic text concatenation requires an algorithm that accepts Arabic characters and then alters or selects the proper shapes of the entered characters according to their positions within a word; then displays the text on screen in a syntactically correct manner [KC91]. Since there are over 90 shapes, the correct shape within the a word needs a complex analysis [KC91]. Not only do we face this huge number of shapes, but also some shapes require double width, for example ‘س’ in its isolated or ending forms.

To illustrate the dependency of the character shape on the position of the character within the word, let us take the character ‘هـ’ (Ha). It has four different shapes:

1. Isolated. For example ‘زمره’.

2. At the beginning. For example ‘هل’.

3. In the middle. For Example ‘سهر’.

4. At the end. For example ‘مدرسه’.

Moreover, Arabic characters in a word are either connectable or not connectable to its successor according to the characteristics of the character. The process of connecting or disconnecting a character requires a complex analysis [KC91]. The Arabic version of MS-Windows provides all the required analyses to display the Arabic text.

The second class of problems comes when numerals are embedded in the text. Arabic or Indian numerals can be written starting from right to left or left to right. Also they can be read starting from right or starting from left [Del86]. Currently, there is no standard for implementing numerals in Arabic computer systems. In some systems, when entering numerals, the cursor stays in one place and the numerals stream off to the left, since numerals run left to right [Smi90].

The third class of problems in displaying text arises when a mixture of Arabic and a European language has to be used. The computer must be able to intermix right-to-left and left-to-right characters and place each character in its proper relative position in the text line. This process is called directional layout. To illustrate the problem of directional layout, let us take an example of an early attempt to mix English with Arabic text in the NCR system [Cas85]. In that system, the English text needs to be reversed. Therefore, the name "AbdulMalik Al-Salman" needs to be keyed into an Arabic document as "namlaS-lA kilaMludbA". Of course, newer systems introduced easier methods for the mixed documents. Examples of bilingual editors on the IBM PC are the Arabic Word Perfect and the Arabic MS Word [Mic93].

Printing the Arabic script needs more effort than displaying the text on the computer screen because it involves some hardware problems such as printer head movement. Printing Arabic scripts also needs a complex analysis, similar to the screen display. Furthermore, printing an Arabic text depends on the type of the printer; the dot matrix printer needs more management than laser printer [kha86]. At the beginning of the last decade, Arabic printers were limited and they also lacked standardization. This fact led to the limitation of using computers in the Arab world. In conclusion, Arabic printers should have the ability to print from right to left, to print also from left to right for bilingual texts, and to perform contextual analysis.

3.3 ARABIC PROGRAMMING LANGUAGES

Experimentation with Arabic programming language environments has been limited and the work on Arabized programming language has been relatively sparse. There are two definitions for Arabization of programming languages [MM87]:

1. Programming language Arabization is to be able to perform input/output in Arabic. This approach is very easy to achieve when the computer screen and printer can deal with Arabic characters. However, it does not enhance the process of computer Arabization.
2. Programming language Arabization is to write programs in Arabic language which ,of course, include the I/O. This approach is followed in this dissertation.

Most of the early programming languages reported in the literature are either a translation of the BASIC programming language or similar in nature to BASIC language. *Al-Khawarizmi* is a BASIC-like programming language that runs under the CP/M operating system [Alkh]. It has control statements, but not iteration statements. *BASIC* صخر (SAKHR) is a translation of the BASIC programming language [Bas95]. It runs under صخر (SAKHR) computers. A third Arabic programming language is Najla [Najla]. It is one of the best translations of BASIC with some enhancements. This programming language is running under AlFarabi computers. One of the problems of the previous Arabic programming languages is they are built for non-popular machines. The fourth Arabic programming language is ARabic BasIc (ARBI) [AYE90]. It is an Arabic programming language that runs under MS DOS. It is compatible with GWBASIC. In ARBI, the Latin characters may appear only in the I/O statements.

Dehlawi and Mandorah listed several programming languages that are similar in nature to Latin BASIC or assembly language [DM86]. The first language is called "غريب"

(GAREB). The name is the abbreviation of the "لغة برمجة عربية للحاسبات" (Arabic Computer Programming Language). It is a subset of the BASIC language; and it was built in AlMousel University, Iraq [DM86]. The second language is called "الخوارزمي" (Al-Kawarizmi). It is different from the one mentioned before. This language was designed to be used by children under 15 years old. It is also a subset of the BASIC language. The third one is an Arabic assembly language. It was designed for a hypothetical machine just to simplify the concept of assembly language to Arab students.

Another programming language that is also related to Arabic was developed by Suleiman and Citrin [SC92, SC95]. The objective of that project is to build a visual programming language that can support any language including Arabic. That project supports the thesis that visual languages are well suited for programmers whose native language is not English. A visual programming language presents most of the syntactic and semantic information in a "BLOX-like" notation (regularized version of flowcharts) [SC95]. The idea of visual programming saves some effort in writing the source code, but it has some drawbacks. It is neither suitable for large programs nor for sophisticated ones that require dealing with records and pointers. Moreover, writing the source code usually takes only a short time compared with the process of debugging the program to make it running. In that project, there was no mention of how the debugging takes place. Finally, it is hard for a person to switch from the visual programming environment to the textual environment when he or she needs to learn more in computer science. We, however, follow the textual environment approach believing that it is more appropriate for the general purpose programming languages.

Finally, a theoretical specification for an Arabic programming language was introduced by El-Affendi [Ela86]. Its structure was translated from Modula, Pascal, and some other languages. It supports five basic types: real, integer, character, boolean, and pointer. Moreover, it has four data structures: array, record, file, and set. There are three control statements: if ... then, repeat ... until, and case statement. Some issues are not defined in that specification. Among these are input/output handling, reserved words, *for* statement, *while* statement, mathematical operators, and built-in functions and procedures. However, it would be a good attempt if it were implemented.

3.4 PASCAL COMPILERS

It is unknown when exactly the first compiler was constructed as the first programming language is undetermined [Sam72]. In general, the concept of high level programming languages started in the early 1950's. Among the oldest programming languages (with their compilers) are FORTRAN, COBOL, and ALGOL 60. Throughout the 1950's and 1960's, compilers were considered very difficult programs to construct. For instance, the first complete FORTRAN compiler took about 18 staff-years to construct [ASU86].

The first known Pascal compiler was developed in 1968 [Wir71]. That compiler was written in FORTRAN. The second compiler was written in Pascal itself in late 1969. After being written, the compiler was then translated by hand into CDC 6000 assembly language and into ICL1900 assembly language [WQ72].The first portable version of Pascal compiler, developed in 1975, is the Pascal- P-compiler [Ber78, PD82]. Rather than producing code for any particular machine, it produces an intermediate code, known as P-code, for a hypothetical stack machine. The P-compiler itself was written in Pascal and compiles a subset of the Standard Pascal. A practical implementation of the P4 Pascal compiler was designed for the Intel 8085A microcomputer [DP80].

The principles and techniques for building compilers have been discussed deeply and widely in many sources [ASU86, Bac79, CJ88, DP80, Eld94, Gri71, Hed75, Mak91, PD82, PP92, Pys88, Wir71, WQ72, Hol90, Hen90]. On the other hand, there are many commercially available compilers for Pascal (usually extended version of Pascal) as well as other programming languages. An example of a Pascal language and compiler is Turbo Pascal [BTP89].

CHAPTER 4

THE APE DESIGN AND IMPLEMENTATION

The APE [Als96] is designed to be an environment analogous to the programming environments in the PCs. To the best of my knowledge, the APE is the first Arabic programming language environment under MS-Windows. Editing, compilation, and execution environments are all under MS-Windows. Editing and compilation windows have been implemented using Visual Basic [Ama94b]. The execution window has been implemented using C language and low-level MS-Windows system calls. The execution window has to be controlled only by the user program, not by the user. In other words, the user should be prevented from altering the window's output. Moreover, once the input is entered, it cannot be changed.

The APE is a multiple-document environment that runs under the Arabic version of MS-Windows. Therefore, the programmer can open more than one file at the same time, but only one file is active at a time. The APE has only one Toolbar and one Statusbar, and all commands are routed to the active file. The highlighted window is the window that has the focus (active window). Figure 4.1 illustrates the main user-windows interactions.

This chapter has five sections. Section 4.1 explains the design of the main window. Section 4.2 shows the various dialog boxes in the APE. Section 4.3 discusses the compilation and execution process. Section 4.4 explains the main APE algorithms. The last section shows some of the APE help windows.

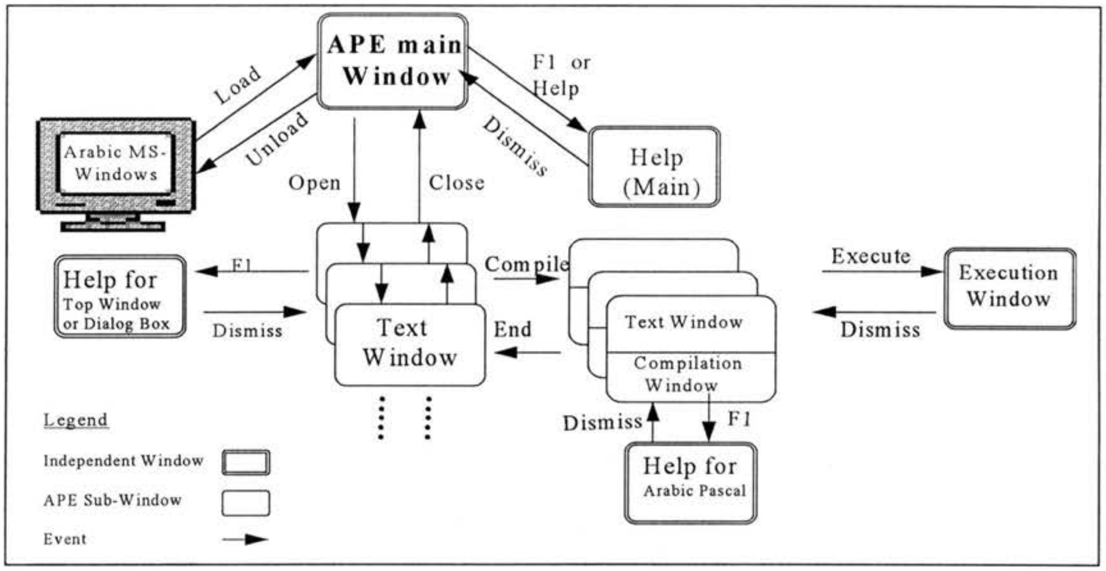


Figure 4.1 Main User-Windows Interactions

4.1 MAIN WINDOW

When the user runs the APE, the main window (Figure 4.2) will show up on the screen. The window is maximized with a single, empty file window. The following subsections explain the various areas in that window.

4.1.1 The software/file title

This area has the software title "باسكال العربي" (Arabic Pascal), followed by the file name. When a user runs the software, the file name will be "بدون_عنوان:١" (Untitled :1). Since the text window is in its maximized mode, the software title and the file name come in the same area. If the text window is not in its maximized mode, the file name will appear only on top of the text window, which is separate from the software title. As provided by the MS-Windows, the "US" icon converts the writing mode to English when clicked. The "ع"

icon converts the writing mode to Arabic. The default mode in the APE is Arabic. If all files are closed, the main window (called MDI Window) will look like Figure 4.3.

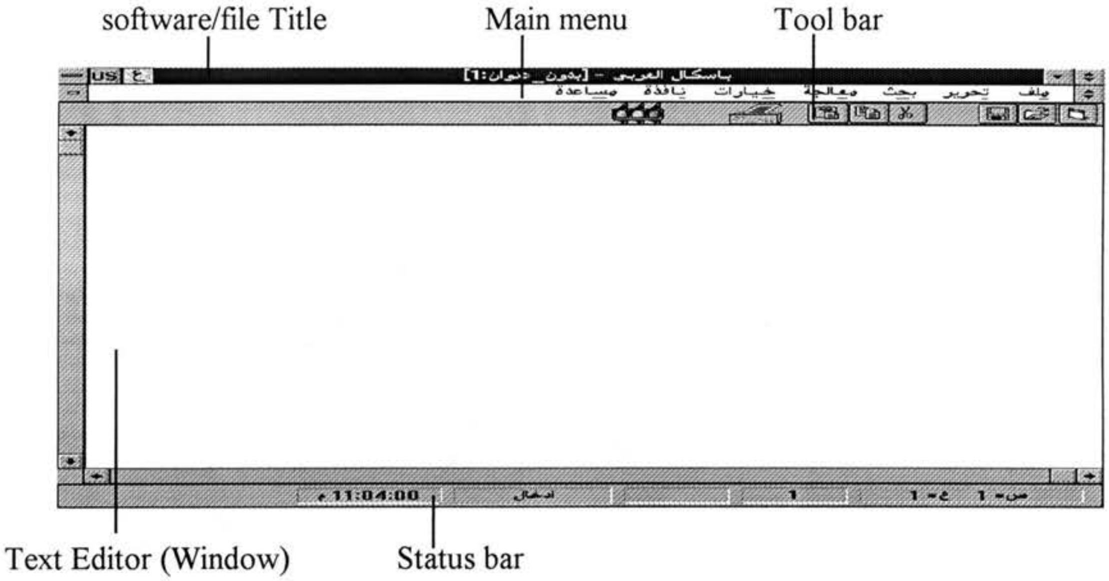


Figure 4.2 A snapshot of the Main Window

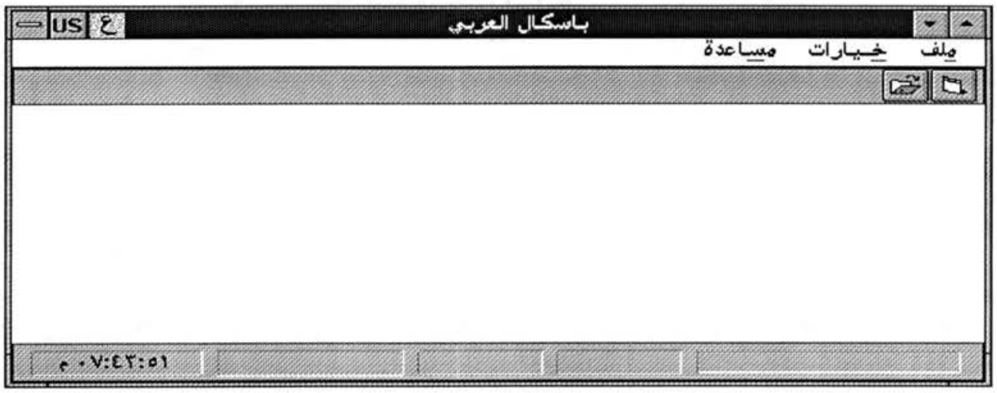


Figure 4.3 The main window after closing all files (MDI Window)

4.1.2 Main Menu

Figure 4.4 (a-f) shows the drop-down submenus in the main window that has at least one file. The following is a translation of these menus (top to bottom). The help menu does not have submenu.

(a) File (New, Open, Close, Save, Save As, Print, Exit, List of the most recently used files).

(b) Edit (Undo, Cut, Copy, Paste, Select All, Date and Time).

(c) Find (Find Only, Find and Replace, GOTO line, GOTO end).

(d) Compilation (Compile, Run, End of Compilation). Section 4.3 gives more explanation about the compilation process.

(e) Options (Toolbar, Statusbar, Font reduction, Font enlargement). The toolbar and statusbar items are used to make the toolbar and the statusbar, respectively, visible or not visible.

(f) Window (Cascade, Tile, Arrange-icons, List of opened files).

(g) Help. (see Section 4.5).

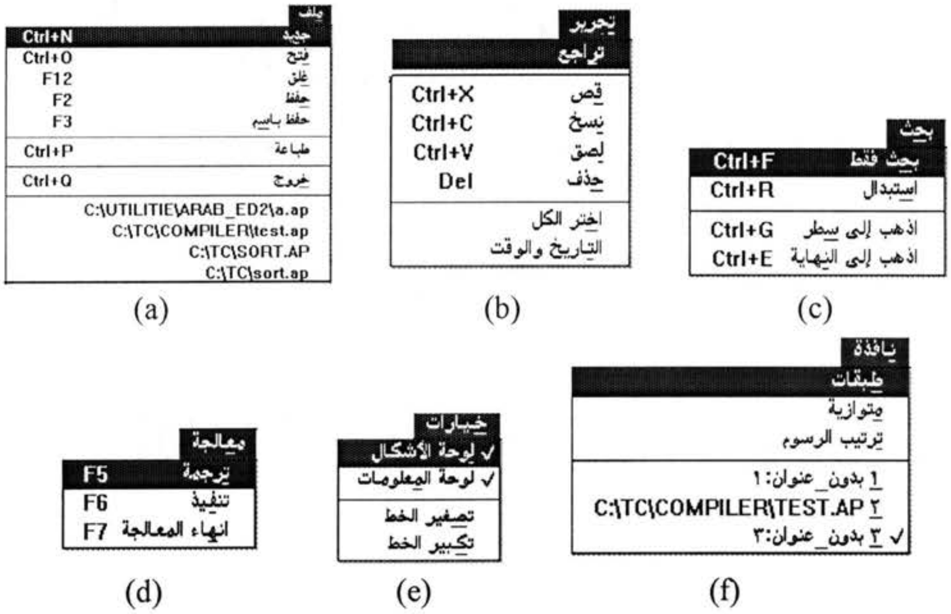
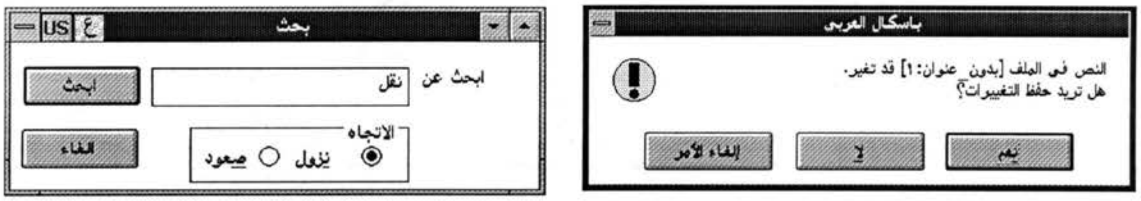


Figure 4.4 Drop-down Submenus

If the main window has no opened files (as shown in Figure 4.3), the main menu consists of only three submenus: file, options, and help. The file submenu has only new, open, exit, and the list of the most recently used files. The options submenu has toolbar and statusbar.

Due to the absence of character set coding standardization, the APE requires the file names to be in English. If the file name is written in Arabic, it may appear as a junk of symbols under the MS-DOS. Also, due to the inconsistency in the Arabic keyboard layout and the absence of the translation of the function keys, the APE uses English short-cut keys, such as F3 and Ctrl+P.

It should be pointed out that all dialog boxes are provided in Arabic including the command buttons. Moreover, all messages are displayed in Arabic. Figure 4.5 (a) shows the Find Dialog Box, and Figure 4.5 (b) shows an example of a message window given to the user when trying to leave a document without saving. The details of the dialog boxes are explained in Section 4.2. The list of possible messages the user can get from the APE is explained in Appendix A.



ابحث عن: نقل

النص فى الملف [بدون_عنوان:١] قد تغير.
هل تريد حفظ التغييرات؟

(a) A Find Dialog Box

(b) Saving file message

Figure 4.5 Examples of a Dialog Box and a message window

4.1.3 Tool bar

It has nine icons. The first three (from the right) are for file operations: open a new file, open an old file, and save the current file. The next three are for editing operations: cut, copy, and paste a selected text. The seventh icon is for printing. The user can print the source program, the compilation output, and the results of running the program. The print command can only print a displayed window. In other words, if a window (source code, compilation output, or result) has been dismissed by the user, the print command cannot print the information that was in that window. The eighth icon is used for compiling command. The last icon (not visible in Figure 4.2) is used for executing a successfully compiled program. The left side of this bar is used for displaying a brief explanation of the tool bar icons and status bar items when the mouse passes over them.

4.1.4 Text Editor

When a new file is displayed on the screen, the cursor will be placed in the rightmost side of the first line. The default mode of writing is Arabic. Since this editor is designed for programming purposes, the APE allows only one font style "Simplified Arabic Fixed." This font has been chosen because all characters have the same width in it, which is recommended in writing programs. While writing, the cursor will move to the left. When the text cursor reaches the left edge of the enclosing window, the cursor stays at the left edge, and scrolls the previous text to the right to accommodate the newly typed characters. Similarly, reaching the bottom line in the window causes an automatic upward scrolling to accommodate the new text. The horizontal scroll bar is used to move the contents of the text window horizontally through the editing area. Note that the indicator of the horizontal scroll bar is placed in the right-most of the bar. Similarly, the vertical scroll bar (in the left side of the editing area) is used to move the contents of the text window vertically through the editing area. The vertical scroll bar is placed to the left hand side to suit the Arabic language nature. Pressing ENTER causes the cursor to jump to the beginning of the next line (rightmost side). It should be pointed out that some keyboard keys work in reverse manner in the Arabic mode. These keys are

a) *Backspace* key deletes the previous character (the character in the right of the cursor),

b) *Home* key returns the cursor to the rightmost character of the current line,

c) *End* key sends the cursor to the last character in the Arabic text (the lower left character), and

d) *Tab* key will cause the cursor to jump eight spaces to the left.

The functions of the other keys remain the same as the English mode.

To edit (cut, copy, etc.) a part of the file, that part has to be selected (highlighted). The selection in the Arabic text should be compatible with the Arabic direction of writing. For example, if the user starts the selection in the middle of a line and drags-down, the left-hand side of the first line should be selected (not the right-hand side as the case in English text).

4.1.5 Status bar

It consists of five items. They are from right to left as follows:

a) Current line number and column number. It indicates the current cursor position. Double clicking on this item causes the GOTO line Dialog Box to appear.

b) Number of lines. It shows the last line number. Double clicking on this item causes the cursor to jump to the end of the file.

c) Modification indicator. This item indicates if the file has been altered or not. If the file is new or not modified since its load time, this item is blank. Whenever a printable character is pressed, this item reflects this change by indicating that the file has been changed "تم تعديله".

d) Insert/Overwrite mode. The default of the Text Editor is insert mode. When the user presses *Ins* key, the mode is flipped.

Time indicator. It displays the current system time in Arabic.

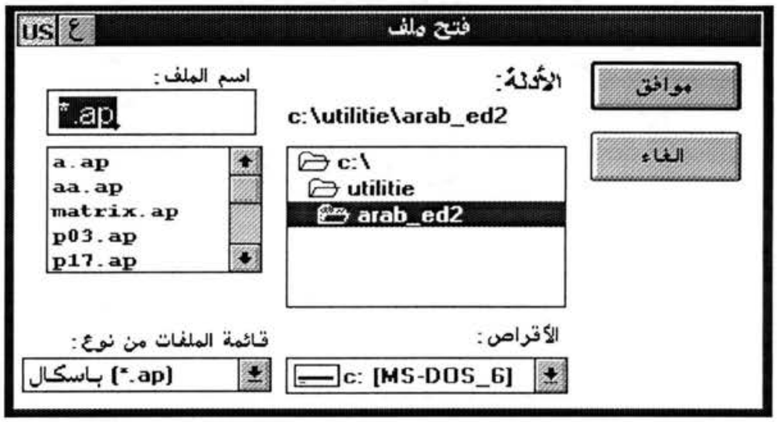
4.2 APE DIALOG BOXES

The APE has five dialog boxes: File, Find, Replace, GoTo line, and Print. All these dialog boxes have to be dismissed before performing any action outside them. Moreover, all these dialog boxes are movable to any place within the Main Window. The default language for all these dialog boxes is Arabic except the File dialog box where the default language is English because the file names should be in English to preserve the compatibility between MS-DOS and MS-Windows.

4.2.1 File Dialog Box

This dialog box is activated when the user wants to perform one of these functions: open, save, or save as. The general format of the File dialog box is shown in Figure 4.6. This figure shows the File dialog box in the case of opening a file. The same dialog box, with different title, is activated when the user wants to save a new file or rename (save as) an old file. Therefore, the title of the dialog box can be:

- Open File (فتح ملف),
- Save File(حفظ ملف), or
- Save As (حفظ باسم).



اسم الملف : *.ap

a.ap
aa.ap
matrix.ap
p03.ap
p17.ap

الأدلة: c:\utilitie\arab_ed2

c:\
utilitie
arab_ed2

قائمة الملفات من نوع: باسكال (*.ap)

الأقراص : c: [MS-DOS_6]

Figure 4.6 File Dialog Box (With Open File Title)

The File dialog box uses the commonly used format for opening and saving files. There are five display boxes and two command buttons.

1. File list box (قائمة الملفات). At the design phase, the APE sets the file pattern to be the list of all files that has the extension of *ap* (Arabic Pascal) within the current directory. The user can choose the needed file by clicking the mouse or using the arrow keys. Double

clicking a file name is equivalent to selecting a file and pressing ENTER or clicking OK (موافق) button.

2. List files of type (قائمة الملفات من نوع). This box controls the file pattern. It has a direct effect on the File list box. The default file pattern is *.ap. The user has the choice to change it to the pattern *.* (list all files within the current directory).

3. Directory box (الأدلة). This box highlights the current directory during the opening of the File dialog box. The user can alter the current directory by clicking the needed directory or using the arrow keys. Changing the current directory automatically changes the File list box content.

4. Drive list box (الأقراص). It displays the available drives in the computer. Changing the current drive causes a change in both directory box and file list box.

5. File name box. This box is editable by the user. The user can write the file name to be opened or saved. The APE offers the user the option of explicitly typing a drive, path, file name, and file pattern. When a file name is typed, it is selected in the File list box. If a pattern is typed, the directory and drive list boxes are updated. Also, the File list box displays the file names which satisfy the pattern.

6. OK (موافق) button. It is the default command button (i.e., pressing ENTER is equivalent to clicking this button). Clicking this button causes the selected or typed file to be opened or saved.

7. Cancel (إلغاء) button. Cancel the action of opening or saving a file. It returns the APE to the status before calling the File dialog box.

4.2.2 Find Dialog Box

This dialog box is activated when the user wants to find a particular word or set of words within the text window. The format of this dialog box is shown in Figure 4.5 (a).

There are two display boxes and two command buttons.

1. Search string box. In this box, the user can type the desired text string. The cursor is placed in the right side of the box for the Arabic text. If the user wants to type an English text, the 'US' button has to be clicked or Alt + left-Shift has to be pressed. If the user selects a specified string from the text window, it will appear automatically in the search string box.

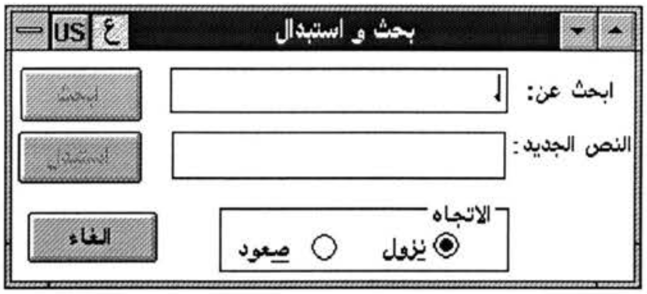
2. Direction box. This box specifies the search direction with respect to the current cursor position, either up or down. The default direction is down. The search will continue in the same direction until the end of the file is reached (in case of down direction) or the beginning of it is reached (in case of up direction).

3. Search (ابحث) button. It is the default command button (i.e., pressing ENTER is equivalent to clicking this button). Clicking this button causes the APE to start searching for the typed string. If the search-string is found in the text window, it will be highlighted. Clicking this button again causes the APE to search for the second occurrence of that string, and so on. If the search-string is not found in the direction specified, a message is displayed to the user.

4. Cancel (إلغاء) button. Cancel the search process. The search dialog box will not terminate until this button is clicked or the left-upper icon is clicked.

4.2.3 Replace Dialog Box

This dialog box is activated when the user wants to find and replace a particular word or set of words within the text window. The format of this dialog box is shown in Figure 4.7.



الاتجاه
● نزول ○ صعود

Figure 4.7 Find and Replace Dialog Box

There are three display boxes and three command buttons.

1. Search string box. The function of this box is similar to the function of the search string box in the Find dialog box.

2. Replace string box. The user may type the text that is going to replace the string in the search string box. If this box is empty that means the user wants to delete the text in the search string box. As in the search string box, the default language is Arabic.

3. Direction box. The function of this box is similar to the function of the direction box in the Find dialog box.

4. Search (ابحث) button. The function of this box is similar to the function of the search button in the Find dialog box.

5. Replace (استبدل) button. This button is inactive (does not respond) until the first occurrence of the search-string is found. When the user clicks this button, the current highlighted text is replaced with the string in the replace string box.

6. Cancel (إلغاء) button. The function of this box is similar to the function of the cancel button in the Find dialog box.

4.2.4 GoTo Line Dialog Box

This dialog box is activated when the user wants to go to a specific line in the text window. The format of this dialog box is shown in Figure 4.8.

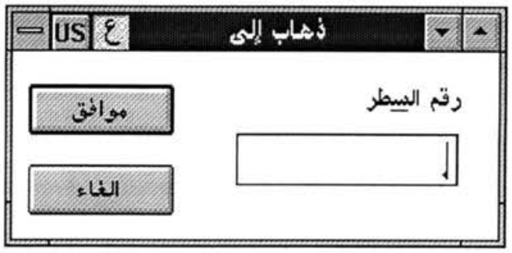


Figure 4.8 GoTo Dialog Box

There are one text box and two command buttons.

1. Line number box. If the user writes a number and presses ENTER (or clicks OK button), the cursor will jump to the beginning of that line number. If the number is invalid (negative number, no number is entered, or a non numeric character), nothing will be performed. If the number is larger than the last line in the text window, the cursor will jump to the last line in the document. Jumping to the first line in the text window can also be accomplished by pressing Ctrl + Home keys.

2. OK (موافق) button. It is the default command button. Clicking this button causes the cursor to jump to the specified line.

3. Cancel (إلغاء) button. It returns the APE to the status before calling the GoTo dialog box.

4.2.5 Print Dialog Box

It is activated when the user wants to print the current source code, compilation output, or the execution results. The format of this dialog box is shown in Figure 4.9.

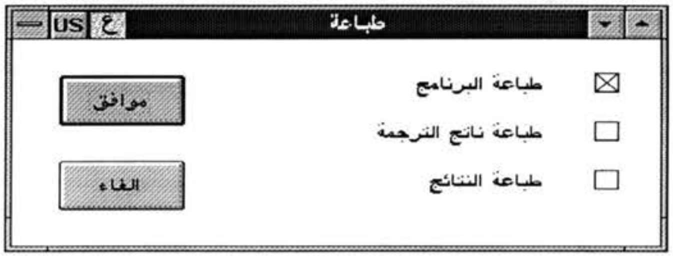


Figure 4.9 Print Dialog Box

There are three check boxes and two command buttons.

1. Program check box. The default of this box is *checked*. When this box is checked, the APE will print the current user program (text window content). The APE prints at the beginning of each page the file name. Each page will have at most 50 lines followed by the page number.

2. Compilation-output check box. The default of this box is *not checked*. When this box is checked, the APE will print the current compilation window content (explained in the following section). The APE prints at the beginning of each page the following phrase نتيجة ترجمة <اسم الملف> (the result of compiling <file name>). Each page will have at most 50 lines followed by the page number. If the compilation window is invisible, the printer prints nothing.

3. Result check box. The default of this box is *not checked*. When this box is checked, the APE will print the current execution window content (explained in the following section). The APE prints at the beginning of the first page the following phrase النتائج (the results). If the execution window is invisible, the printer prints nothing.

4. OK (موافق) button. It is the default command button. Clicking this button causes the attached printer, if any, to start printing the checked boxes.

5. Cancel (إلغاء) button. It returns the APE to the status before calling the Print dialog box.

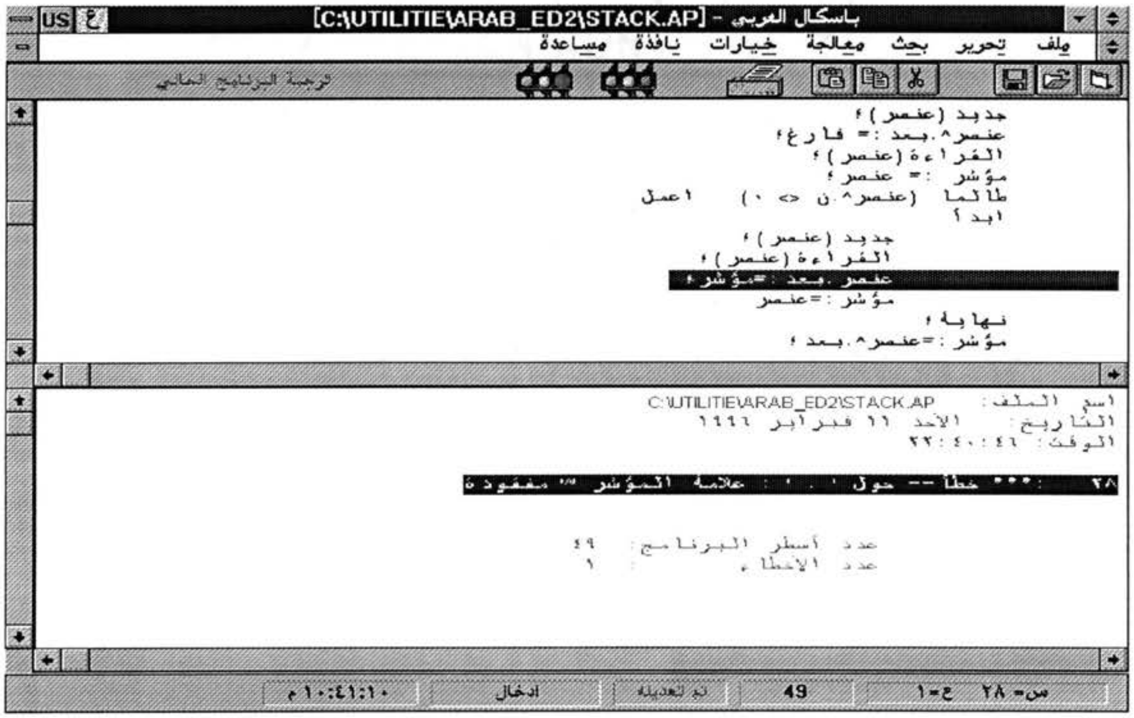
4.3 COMPILATION AND EXECUTION PROCESS

When a programmer compiles a program, the APE window splits into two windows, the program window and the compilation window (display window). The compilation window contains the error messages, if any, and some other helpful information such as time of compilation, number of lines compiled. During the compilation process, a yellow traffic light icon turns on to indicate that the compilation process is in progress. If there is no error, the green light will be on. In case of errors, the red light will be on. There is a direct connection between each error message and its corresponding line in the source code that produces the error. Figure 4.10 depicts the APE windows after an unsuccessful compilation. The highlighted lines are the statement in the source code that causes the error and the corresponding error message. Note that the text color of the compilation window is red indicating that some errors has been encountered. Note also that the traffic light of the execution icon is turned red. The APE also supports interactive modification of the source code. In other words, the programmer can switch from the compilation window to the source code window easily without loosing any of these windows. Switching from a window to another can be accomplished by pressing the *tab* key. Figure 4.11 shows the APE windows after a successful compilation. Note that both the text color of the compilation window and the execution traffic icon are in green.

When a program is error-free, it becomes ready for execution. the APE creates a new window for execution when the user executes a program. The default language in the execution window is Arabic. If a run-time error occurs, an error message will be displayed and the execution will be aborted. As long as the source code is not altered, the programmer can execute the program as many times as necessary without the need for recompilation. Figure 4.12 shows the execution window after the end of an execution process. The program shown partially in this figure creates a linked list and then displays its elements as first-in last-out. Pressing F7 (End of Compilation) dismisses the compilation and execution windows and returns the program window to its original size.

The execution window and the Print dialog box are designed by the APE to be the top-most windows in the system. Therefore, both windows will be on top of all other windows even if the user switches to another software. The title of the execution window is the name of the executable file. When the execution terminates, the window will be locked (the window content cannot be altered) and its title will indicate the termination of the execution.

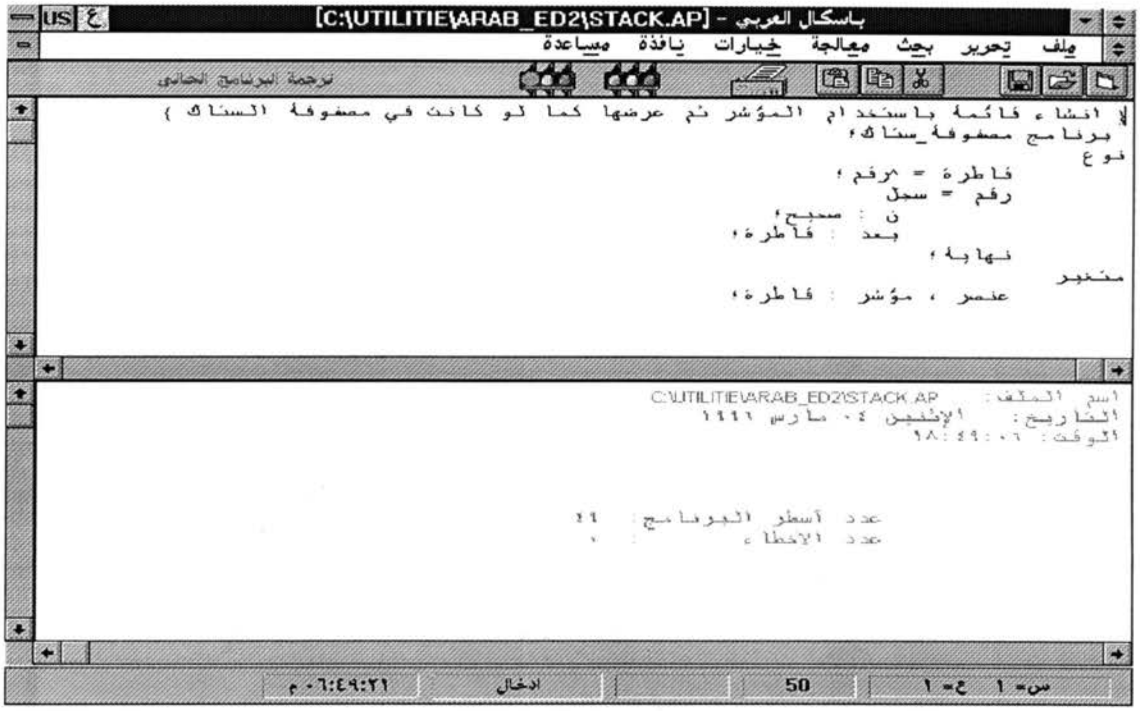
If a user program has neither a read from the standard input nor a write to the standard output, the execution window will not show up. Also if there is read from the standard input and the output is to be sent to a file, the execution window has to be dismissed before the output file can be opened. When the execution window is dismissed, the buffered output is transferred to the output file.


US ع
[C:\UTILITIE\ARAB_ED2\STACK.AP] - باسكال العربي
ملف تحرير بحث معالجة خيارات نافذة مساعدة
ترجمة البرنامج الحالي
١٠:٤١:١٠ م
ادخال
تم تعديله
49
ع=١
س= ٢٨

```
جديد (عنصر) ؛
عنصر^.بعد := فارغ؛
القراءة (عنصر) ؛
مؤشر := عنصر ؛
طالما (عنصر^.ن <> ٠) اعمل
ابدأ
    جديد (عنصر) ؛
    القراءة (عنصر) ؛
    عنصر .بعد :=مؤشر ؛
    مؤشر :=عنصر
نهاية ؛
مؤشر :=عنصر^.بعد ؛
```

```
اسم الملف :      C:\UTILITIE\ARAB_ED2\STACK.AP
التاريخ :      الأحد ١١ فبراير ١٩٩٦
الوقت : ٢٢:٤٠:٤٦

٢٨ : *** خطأ -- حول '.' : علامة المؤشر '^' مفقودة

عدد أسطر البرنامج : ٤٩
عدد الاخطاء       : ١
```

Figure 4.10 A snapshot of the compilation window (unsuccessful compilation)

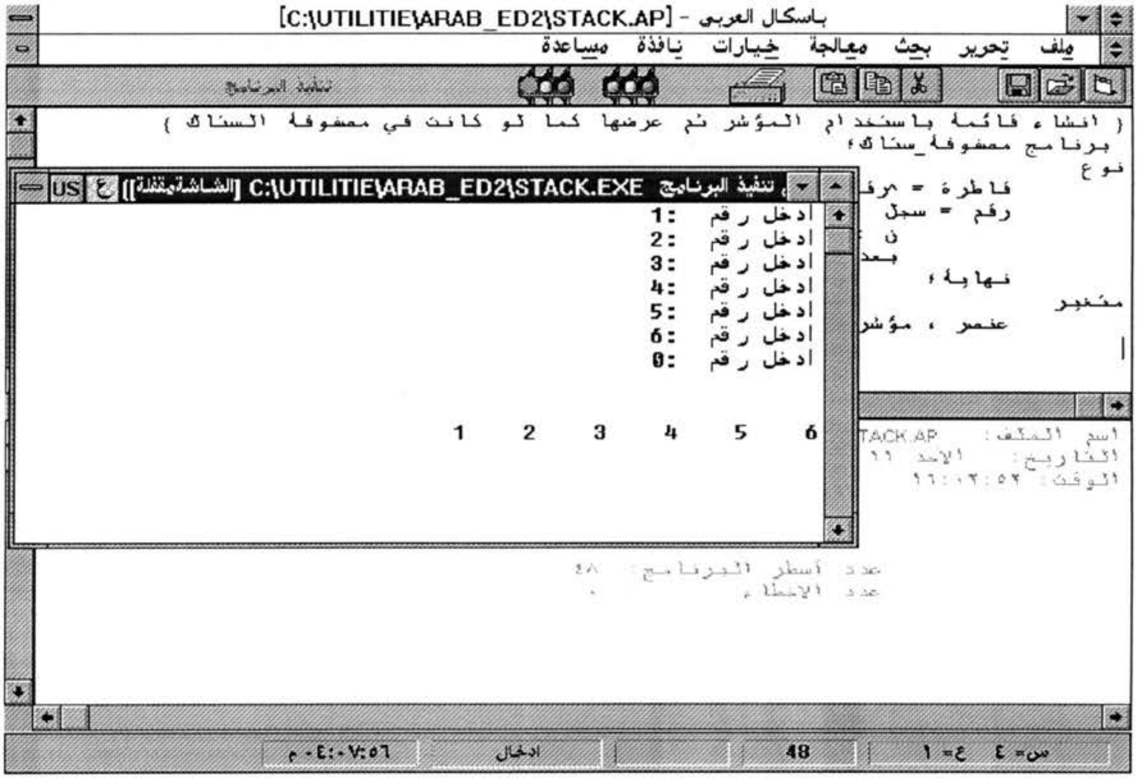

[C:\UTILITIE\ARAB_ED2\STACK.AP] - باسكال العربي
ملف تحرير بحث معالجة خيارات نافذة مساعدة
ترجمة البرنامج الحالي

```
{ انشاء قائمة باستخدام المؤشر ثم عرضها كما لو كانت في مصفوفة السناك }
برنامج مصفوفة_سناك؛
نوع
   قاطرة = ^رقم؛
   رقم = سجل
         ن : صحيح؛
         بعد : قاطرة؛
   نهاية؛
متغير
   عنصر ، مؤشر : قاطرة؛
```

```
اسم الملف : C:\UTILITIE\ARAB_ED2\STACK.AP
التاريخ :   الإثنين ٠٤ مارس ١٩٩٦
الوقت : ١٨:٤٩:٠٦

عدد أسطر البرنامج : ٤٩
عدد الاخطاء       : ٠
```

س= ١ ع= ١
50
ادخال
٠٦:٤٩:٢١ م

Figure 4.11 A snapshot of the compilation window (successful compilation)


[C:\UTILITIE\ARAB_ED2\STACK.AP] - باسكال العربي
ملف تحرير بحث معالجة خيارات نافذة مساعدة
تنفيذ البرنامج
[الشاشة مقفلة] C:\UTILITIE\ARAB_ED2\STACK.EXE تنفيذ البرنامج

```
{ انشاء قائمة باستخدام المؤشر ثم عرضها كما لو كانت في مصفوفة السناك }
برنامج مصفوفة_سناك؛
نوع
```

```
ادخل رقم  1:
ادخل رقم  2:
ادخل رقم  3:
ادخل رقم  4:
ادخل رقم  5:
ادخل رقم  6:
ادخل رقم  0:

  6     5     4     3     2     1
```

```
اسم الملف : TACK.AP
التاريخ :   الاحد ١١
الوقت : ١٦:٠٢:٥٢

عدد أسطر البرنامج : ٤٨
عدد الاخطاء       : ٠
```

س= ٤ ع= ١
48
ادخال
٠٤:٠٧:٥٦ م

Figure 4.12 The APE including the execution window

4.4 APE ALGORITHMS

The APE has been developed using Visual Basic which is an event-driven programming language. Figure 4.13 illustrates the general software architecture of the APE. A dialog box in the figure is a special window that prompts the user to supply information needed by the software to continue with the application. A handler is a piece of code that performs some actions in response to some events. As shown in the figure, there are mainly ten handlers that control the APE functions. The following subsections describe the algorithms and actions of these handlers. The last subsection (4.4.11) sketches the algorithm for controlling the execution window

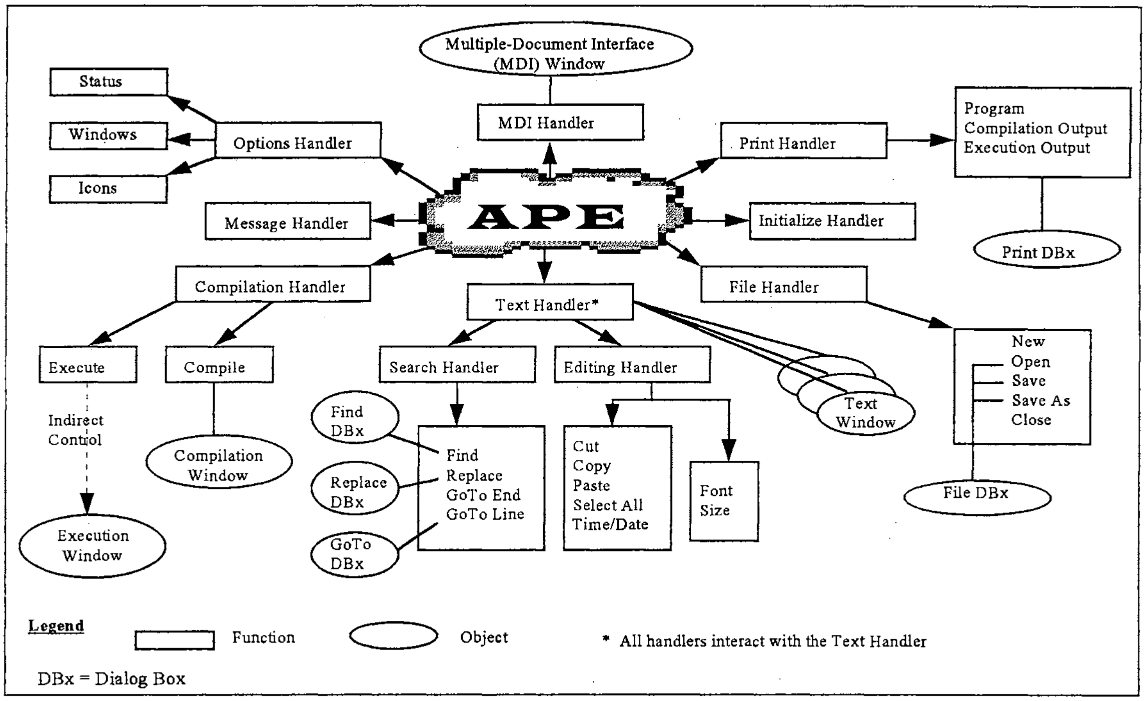


Figure 4.13 Software Architecture of the APE

4.4.1 MDI handler

The MDI handler is the handler that is responsible for all actions to be taken in response to the main window events. The MDI handler also handles the events for all the objects included in the MDI window such as toolbar icons, statusbar items, and the menu items that are available when there is no file opened (as in Figure 4.3). Table 4.1 describes the actions related to the MDI window.

Table 4.1 Actions related to the MDI Window

Object	Event	Action
MDI window	load	show the MDI window, call Initialize handler, get the most-recently-used-files, and open a new text window.
MDI window	unload	close/save all opened files (text windows) and quit the Help Window, if any.
New-icon and menu-new	click	call File new (see File handler)
all icons in the toolbar and all items in status bar	move over	display a help message
open-icon and menu-open	click	call File open (see File handler)
save-icon	click	call Save file (see File handler)
cut-icon	click	call Cut procedure (see Editing handler)
copy-icon	click	call Copy procedure (see Editing handler)
paste-icon	click	call Paste procedure (see Editing handler)
print-icon	click	show the Print dialog box
compile-icon	click	call Compile procedure (see Compilation handler)
run-icon (red or yellow lights)	click	display a correction message
run-icon (green)	click	call Execute procedure (see Compilation handler)
menu-exit	click	perform Window-Close function (system command)
status-position	double-click	show the GoTo dialog box
status-size	double-click	move the cursor to the end of the text window
menu-option-toolbar	click	flip the visibility of the toolbar
menu-option-statusbar	click	flip the visibility of the statusbar
menu-recent-files	click	call opening-file (see File handler)
menu-help	click	open the main software help window (contains information about the APE and the Arabic Pascal)
---	F1 key	display the APE help window
toolbar	resize	call Options handler to adjust the icons according to the new dimensions
statusbar	resize	call Options handler to adjust the items according to the new dimensions

4.4.2 Initialize Handler

Most of the initializations to dialog boxes take place during the design phase. As it is known in Visual Basic [Ama94b], any window or dialog box can be initialized to specific size, place, and attributes during a so-called 'design mode'. Therefore, the APE initial windows are designed by choosing the appropriate properties. However, the initialize handler performs the necessary initializations for non-dialog box related issues. These issues are:

Defining the global constants

Declaring the MS-Windows library functions and procedures that are used by the APE

Converting the keyboard to the Arabic mode

Setting the following:

the working directory to the directory containing the APE

insert/overwrite mode to insertion

cursor position to line one column one (right side)

Software title to 'باسكال العربي' (Arabic Pascal)

font size to 12

4.4.3 File Handler

The File handler deals with all activities related to files including the File dialog box. The procedures that handle the actual file opening, saving, and checking are as follows:

1. Open-file procedure

show the File dialog box

get the file name from the File dialog box

```
unload File dialog box
if the file exists call opening-file procedure (see next)
update the most-recently-used-files to include this file
```

2. Opening-file procedure

```
open the file for reading and writing and load it in a text window
if the file size is over 32k, display an error message to the user
make all icons visible to the user
set the cursor in the upper-right corner
set the line and column numbers to 1
calculate the file length (in terms of lines) and display it in the status bar
```

3. New-file procedure

```
show a text window titled as "untitled : index"
make all icons visible to the user
set the line and column number to 1
```

4. Save-file procedure

```
if the file has no name (untitled) show the File dialog box and get a file name from the user
if file name already exists, display a warnning message to the user
write the text window content in the file
update the most-recently-used-files to include this file name
```

The actions related to the File dialog box are described in Table 4.2

Table 4.2 Actions related to the File Dialog Box

Object	Event	Action
file-type box	click	select either *.* or *.ap update the file name box and file list accordingly
directories box	change	propagate directory changes to current directory path and file list
drives box	change	propagate drive change to current directory path and

		file list
file list box	click	echo the selected name in the file name box
file list box	double-click	echo the selected name in the file name box and do as OK click
file name box	change	if empty then disable the OK button
---	F1 key	display the file help window
OK button	click	if incorrect name then send a message else return the file name
Cancel button	click	hide the File dialog box
File dialog box	load	set the keyboard language to English
File dialog box	unload	set the keyboard language back to Arabic

4.4.4 Text Handler

The main function of the Text handler is to control the events that happen in the text window. The Text Handler has control over the Editing Handler and the Search Handler which are specialized handlers that deal with text editing and search activities within the text window, respectively. Table 4.3 describes the actions taken when an event occurs in the text window.

Table 4.3 Actions related to the Text window

Object	Event	Action
text window	resize	adjust the text window to the new size
text window	unload	close the active text window (save the file, if needed). If all text windows are closed, make all icons related to text window invisible, and refresh the statusbar (the APE window will looks like Figure 4.3)
menu-close	click	same as event 'unload' above
menu-save	click	same as save-icon
menu-save-as	click	show the File dialog box
menu-print	click	same as print-icon
menu-undo	click	undo the last editing action
menu-cut	click	same as cut-icon
menu-copy	click	same as copy-icon
menu-paste	click	same as paste-icon
menu-delete	click	delete the last keyed character or the highlighted text
menu-select-all	click	call Editing handler
menu-date	click	call Editing handler
menu-find	click	call Search handler
menu-replace	click	call Search handler
menu-Goto-line	click	call Search handler
menu-Goto-end	click	let cursor jump to the end of the text window

menu-compile	click	same as compile-icon
menu-run	click	same as run-icon (green and red)
menu-compilation-end	click	hide the run-icon, hide the compilation window, restore the original font size for text window, and perform text-window-resize
menu-reduce-font	click	call Editing handler
menu-enlarge-font	click	call Editing handler
menu-cascade	click	call Options handler
menu-tile	click	call Options hander
menu-arrange	click	call Options handler
menu-list-of-opened-files	click	bring to front (to be the active text window) the chosen file
---	F1 key	display the APE help window
text window	change	modify the file status (in statusbar) to 'Modified' modify the file size (number of lines), if needed.
text window	got focus	calculate the current line number and the current column number and display that in the statusbar. Calculate the file size and display that in the statusbar
text window	key down	(used for control keys) if *Ins* key is pressed modify the statusbar
compilation window	got focus	if the text color is green (no errors), then there is no correspondence between the text window and the compilation window else if the line starts with a number then highlight it and its corresponding line in the text window
compilation window	key up	ignore all keys except ENTER and TAB, if ENTER do as got focus if TAB switch from/to the text window

4.4.5 Editing Handler

This handler does the actual editing functions. There are two types of editing functions: functions related to selected text or insertion in current position (namely cut, copy, paste, select all, and time/date) and functions related to the entire text window (namely font size).

1. Cut: *copy selected text to Clipboard and delete selected text from the text window*

2. Copy: *copy selected text to Clipboard*

3. Paste: *place text from Clipboard into the current position in text window*

4. Select all: *highlight all the text in the text window*

5. Time/date: *insert the current date and time in the current cursor position*

5. Reduce font: *reduce the font size by 1*

6. Enlarge font: *increase the font size by 1*

4.4.6 Search Handler

The search handler takes care of searching activities. There are three dialog boxes related to this handler: Find, Replace, and GoTo line. Tables 4.4, 4.5, and 4.6 describe the actions related to these dialog boxes, respectively. The main algorithm of the search handler is as follows:

```
case   find:
       if there is a highlighted text in the text window, copy it in the
               search-string
       else copy the previous search-string, if any, in the current
               search-string
       show the find dialog box
       call find-string (performed if the user clicks search, see next)
case   replace:
       if there is a highlighted text in the text window, copy it in the
               search-string
       else copy the previous search-string, if any, in the current
               search-string
       show the replace dialog box
       call find-string (performed if the user clicks either search or
               replace, see next)
case   GoTo line:
       show the GoTo line dialog box (the action is taken within the
               dialog box, see table 4.6)
```

The algorithm for find-string is as follows:

```
if the direction of search is down (from the cursor position and downward)
        copy the text from the current cursor position to the end of the text
                window into a temporary string
        find the first occurrence of the search-string
else    copy the text from the current cursor position to the beginning of
                the text window into a temporary string
        find the last occurrence of the search-string
if the search-string is not found display a message to the user
else    highlight the found text in the text window
        move the cursor to the end of that occurrence
```

Table 4.4 Actions related to the Find Dialog Box

Object	Event	Action
Find dialog box	load	make find button disabled and the search direction down
direction	click	flip the direction of search
---	F1 key	display find help window
find button	click	read the search-string, then call find-string
cancel button	click	save the text in the search-string box and unload the find dialog box
search-string	change	if the search string is empty, disable the find button, else enable it

Table 4.5 Actions related to the Replace Dialog Box

Object	Event	Action
Replace dialog box	load	make find and replace buttons disabled, and the search direction down
direction	click	same as in find dialog box
---	F1 key	display replace help window
find button	click	read the search-string, then call find-string, and enable the replace button
replace button	click	cut the original text, read the replace-string, if any, paste the replace-string, then call find-string
search-string	change	same as in find dialog box

Table 4.6 Actions related to the GoTo line Dialog Box

Object	Event	Action
OK button	click	read the line-number if valid number (> 0) move the cursor to the beginning of the specified line unload the GoTo dialog box
Cancel button	click	unload the GoTo dialog box
---	F1 key	display goto help window

4.4.7 Print Handler

The print handler is the piece of code that is responsible for printing the source code, the compilation output, and the execution results. There is only one dialog box related to this handler. Table 4.7 describes the actions related to that dialog box. The main algorithm of the print handler is as follows:

```
case    print the source code (user program):
        copy the text window content in a temporary file for formatting
        the page-header gets the file name (to be printed on top of
                all pages)
        call do-printing (see below)
case    print the compilation output:
        if there is no compilation window, display an error message and
                then exit
        copy the compilation window content in a temporary file for
                formatting
        the page-header gets the following string:
                نتيجة ترجمة <اسم الملف>    ('Compilation of' <file name>)
        call do-printing (see below)
case    printing the execution result:
        if there is no execution window, display an error message
```

```
            and then exit
        copy the execution window  content in a temporary file for
            formatting
        the page-header gets the string النتائج ('The Results')
        call do-printing (see below)
```

The algorithm for *do-printing* is as follows:

```
    while there is a line
        copy the line in a temporary buffer
        if top of the page, print the page-header
        copy the line to the printer buffer
        if bottom of the page, print the page number centered and
            increment the page number
        if last line, print empty lines until the page end then print the page
            number
    send print command to the printer
```

Table 4.7 Actions related to the Print Dialog Box

Object	Event	Action
Print program-check-box	click	flip the program-check-box (enabled/disabled)
Print compilation-check-box	click	flip the compilation-check-box (enabled/disabled)
Print result-check-box	click	flip the result-check-box (enabled/disabled)
OK button	click	start printing the checked boxes
---	F1 key	display print help window
Cancel	click	unload the Print dialog box (do not print)

4.4.8 Options Handler

This handler is responsible of keeping track of the statusbar, the opened windows, and the icons, if any. Table 4.8 describes the action taken for each request received by the option handler.

Table 4.8 Actions taken by the Option handler

Request	Action
Cascade windows	make a system call to cascade all available text windows
Tile windows	make a system call to tile all available text windows
Arrange minimized windows	make a system call to arrange all icons
resize (window max. to normal) the MDI window	if the toolbar is visible, shift all the toolbar icons left to fit in the normal window. Do the same for all items in the statusbar, if it is visible. If there is a text window, resize it.
resize (from normal to max.) the MDI window	if the toolbar is visible, shift all the toolbar icons right to start from the right corner in the maximized window. Do the same for all items in the statusbar, if it is visible. If there is text window, resize it.

4.4.9 Message Handler

Whenever a message needs to be sent to the user, this handler is called. This handler is responsible for displaying the message in Arabic (including the command buttons) with right alignment. Messages can be classified based on their content into three types: error messages, informative messages, and dialog messages. The first two types require only an OK button. For dialog messages, the message handler displays two (yes or no) buttons or three (yes, no, and cancel) buttons depending on the message content. Moreover, the message handler may include either an exclamation icon or a question icon within the message. The message handler takes the user response and passes it to the caller to take the appropriate actions. Appendix A lists all APE messages.

4.4.10 Compilation Handler

The compilation handler is the mediator between the APE and the Arabic Pascal Compiler (APC). This handler has two main functions: compilation process and execution process.

1. Compilation process

```
disable the run-icon, if it was enabled
turn on the yellow signal indicating the compilation progress
split the main window into two windows: text window (reduce its font size)
        and compilation window
set the compilation window as a read-only window
if the file (text window) was not saved, save it
call the APC to compile the current file
if there is no error, set the color of the compilation window text to green
        and turn on the green traffic light.
else set the color of the compilation window text to red and turn on the red
        traffic light
display the APC output
if no errors (arrange the program for execution)
        call the Turbo Assembler to assemble the program
        call the Turbo Linker to link the program
        delete the assembly and object files
else    highlight the first error in compilation window and its
                corresponding line in the text window
```

2. Execution process

```
find the executable file name that corresponsds to the active text window
execute the program (the execution window handler is responsible for
        creating and managing the execution window)
```

4.4.11 Execution window handler

As stated earlier the execution window has to be controlled only by the user program. Therefore, this handler has to make the cursor movement based on the user program. For example, if the user program has some input, the cursor should be shown, otherwise no cursor is shown. If a user types mistakenly some input and then erases it, it should be removed from the input buffer. There are many functions that are handled by this handler, the main ones are:

1. Show cursor

```
create the caret (cursor)
if the cursor reaches the left side of the window, move it to the next line at the right side of the window
else decrement the cursor position by 1 (move to the left for Arabic)
```

2. Write text

```
if the user text is longer than the window zise, break the line into sublines (to control the Arabic display)
count the line characters to move the cursor accordingly
display the text
```

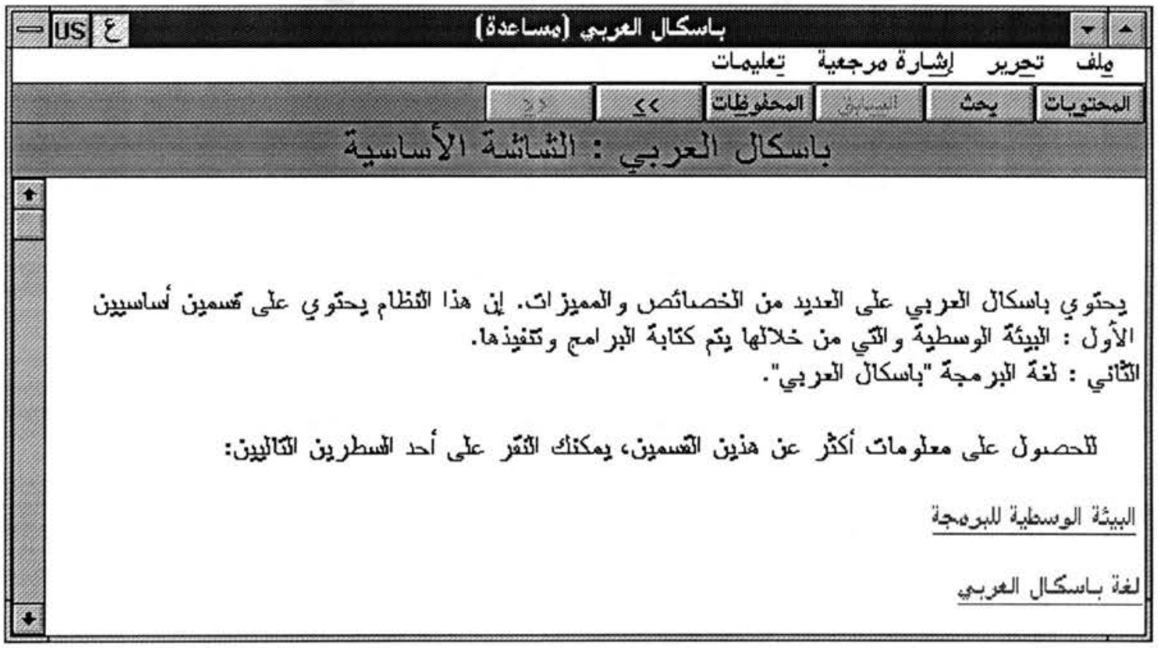
3. Read text

```
wait for the user input
keep track of the user input:
    if backspace    erase the last character from the input buffer and adjust the cursor
    if ENTER        accept the input buffer and move the cursor to the next line (right side)
    if TAB          move the cursor 8 spaces
```

```
          if Control keys      ignore
          else                 insert in the input buffer
4. Paint window  (required when the execution window is resized or scrolled [Cro94]])
     find the area in the execution window that has been altered
          (need to be re-painted)
     refresh the window
     redisplay the text
5. Initialize window
     create the window (titled with file name)
     show the cursor if there is input
     open a text-file to capture the user program input and output
          (used when printing the result)
6. Terminate execution
     close the text-file
     change the window title to indicate the termination of execution
     destroy the caret (cursor)
```

4.5 HELP WINDOWS

In general, the creation of the on-line help is separate from the MS-Windows application and it can be invoked as a stand-alone application. After the creation of on-line help, it can be attached to the software. The APE provides an on-line help file that explains the necessary information needed for dealing with the system. When the user clicks the help menu, the main help window is displayed. This window (Figure 4.13) gives an introduction of the system and leads to two other windows: the APE interface help window (Figure 4.14 shows part of this window) and the Arabic Pascal help window (Figure 4.15 shows part of this window). When the user presses F1 key, an explanation about the top most window will be displayed. If, say, the main window is shown (and there is no dialog boxes), the APE interface help window will be displayed. If the user opens the search dialog box and presses F1 key, a help window will show up to explain the search dialog box, and so on.



باسكال العربي : الشاشة الأساسية

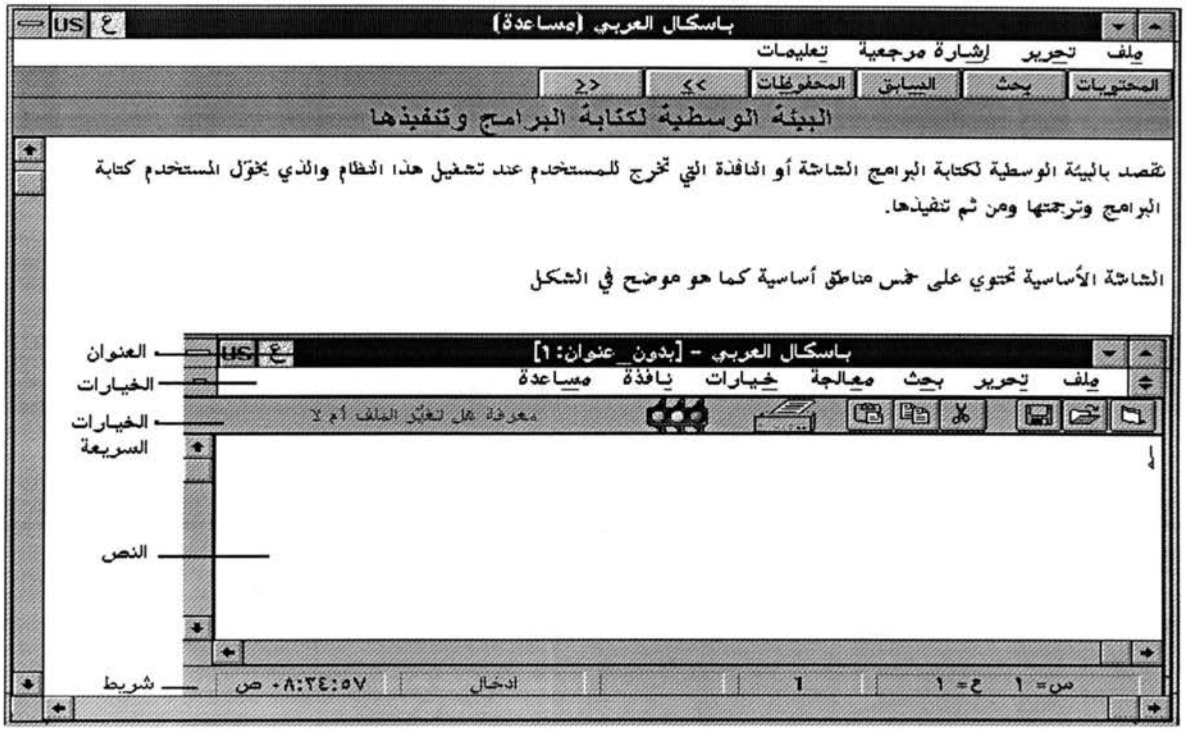
يحتوي باسكال العربي على العديد من الخصائص والمميزات. إن هذا النظام يحتوي على قسمين أساسيين
الأول : البيئة الوسطية والتي من خلالها يتم كتابة البرامج وتنفيذها.
الثاني : لغة البرمجة "باسكال العربي".

للحصول على معلومات أكثر عن هذين القسمين، يمكنك النقر على أحد السطرين التاليين:

البيئة الوسطية للبرمجة

لغة باسكال العربي

Figure 4.14 The main help window

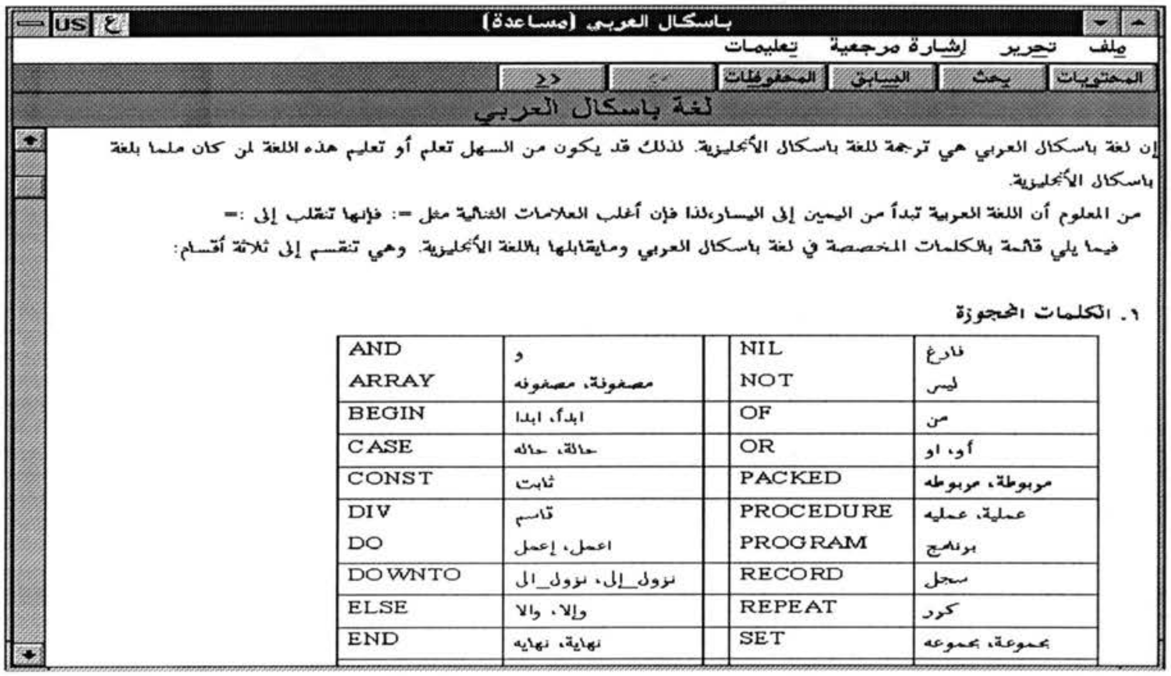


البيئة الوسطية لكتابة البرامج وتنفيذها

نقصد بالبيئة الوسطية لكتابة البرامج الشاشة أو النافذة التي تخرج للمستخدم عند تشغيل هذا النظام والذي يخوّل المستخدم كتابة البرامج وترجمتها ومن ثم تنفيذها.

الشاشة الأساسية تحتوي على خمس مناطق أساسية كما هو موضح في الشكل

Figure 4.15 Partial display of the APE interface help window



لغة باسكال العربي

إن لغة باسكال العربي هي ترجمة للغة باسكال الأنجليزية. لذلك قد يكون من السهل تعلم أو تعليم هذه اللغة لمن كان ملما بلغة باسكال الأنجليزية.

من المعلوم أن اللغة العربية تبدأ من اليمين إلى اليسار،لذا فإن أغلب العلامات الثنائية مثل =: فإنها تنقلب إلى :=

فيما يلي قائمة بالكلمات المخصصة في لغة باسكال العربي ومايقابلها باللغة الأنجليزية. وهي تنقسم إلى ثلاثة أقسام:

١. الكلمات المحجوزة

AND ARRAY	و مصفوفة، مصفوفه		NIL NOT	فارغ ليس
BEGIN	ابدأ، ابدا		OF	من
CASE	حالة، حاله		OR	أو، او
CONST	ثابت		PACKED	مربوطة، مربوطه
DIV DO	قاسم اعمل، إعمل		PROCEDURE PROGRAM	عملية، عمليه برنامج
DOWNTO	نزول_إلى، نزول_الى		RECORD	سجل
ELSE	وإلا، والا		REPEAT	كرر
END	نهاية، نهايه		SET	مجموعة، مجموعه

Figure 4.16 Partial display of the Arabic Pascal help window

CHAPTER 5

DESIGN AND IMPLEMENTATION OF THE ARABIC PASCAL COMPILER

A compiler is a computer program that accepts as input the text of a program expressed in a given programming language (source code) and produces an equivalent program in another language (assembly code or object code) [Mak91]. Figure 5.1 shows the main parts of the Arabic Pascal Compiler (APC) with their most common interfaces.

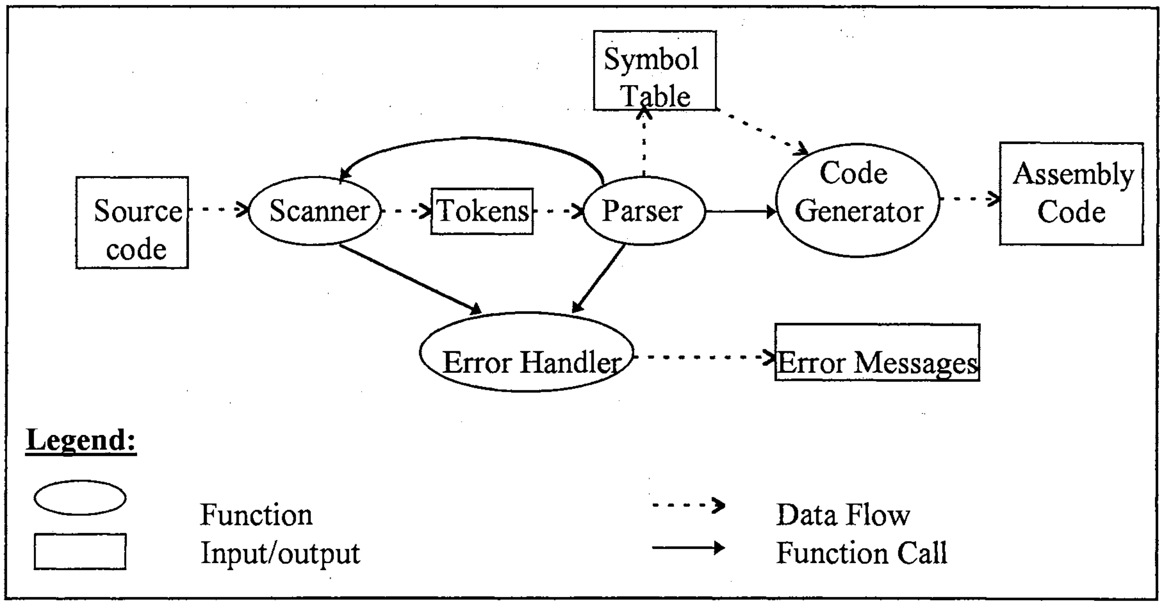


Figure 5.1 Organization of the Arabic Pascal Compiler

As shown in Figure 5.1, there are four essential functions: the scanner, the parser, the code generator, and the error handler. The basic function of the scanner is to read the source text file as a string of characters and to recognize from it a stream of words, numbers, and symbols. Each word, number, or symbol is called a *token* [CJ88]. Details of scanner functions are discussed in Section 5.1. The fundamental task of the parser is to take the stream of tokens as input, build symbol tables, analyze the source program based on the syntax, check the correctness of different types, and finally call the code generator. Section 5.2 is devoted to the description of the parser. The final segment in the APC is the code generator. It generates an equivalent assembly code for the source program. Explanation of the code generator is presented in Section 5.3. During the process of compilation, if an error occurs the error handler is invoked. More explanation of the error handler is in Section 5.4. The last section, Section 5.5, gives examples of two Arabic Pascal programs along with their equivalent English Pascal programs.

The APC is developed in C and compiled using Turbo C. The compiler generates 8086 assembly code. The generated code is assembled and linked using Turbo Assembler and Turbo Linker [BTA93]. Therefore, the targeted machine is IBM PC or compatibles. We chose this machine because it is the most widely used machine at the personal level. The compiler requires an Arabic interface to write and edit the Arabic Pascal programs.

Before starting the explanation of various parts and algorithms of the APC, we should point out that almost all examples and reserved words mentioned in this chapter are from the Standard English Pascal [Coo83, Tib81, WH86]. Appendixes B-E are devoted to explain the structure and the terms in the Arabic Pascal language.

5.1 THE SCANNER

Most of the compiler runtime is spent on lexical analysis in the scanner [PP92, p.48]. The scanner reads line by line from the input file, reducing each line to manageable tokens (words, numbers, strings, and special symbols). It is therefore incumbent upon a compiler designer to exert some effort in making the scanner efficient. Since regular grammar is sufficient to fully define all different types of tokens in the Arabic Pascal, a finite-state automaton is adequate to implement the scanner. The scanner performs four major tasks:

1. Removing extraneous blanks, carriage returns, and new-line characters.
2. Removing comments.
3. Identifying tokens.
4. Reporting errors it discovers by calling the error handler.

There are two common approaches for the compilation process. The first approach performs the compilation in multi-phases. Each phase is completed for a given source program before the next phase starts. This approach is called *multi-pass compilation*. An example of this approach is the *two-pass compilation* which has two scan phases. In the first scan, the compiler resolves all forward references. The second scan does the rest of the compilation. The second approach interleaves or carries out in parallel the phases of compilation in one step. It is called *one-pass compilation*. This approach solves the problem of forward references by leaving holes in the code where the address needs to appear. When the target address becomes known, the compiler returns to the output code and patches the offset or the branch address with the correct value. This method is called *backpatching* [Pys88]. The APC uses the latter approach because it better suits block-structured languages.

Unlike the usual scanning, the Arabic Pascal scanner scans the source code from right to left instead of from left to right. The scanning should be done in this way because the Arabic writing and reading are from right to left. Moreover, the Arabic Pascal scanner does not have the problem of upper and lower cases because these are not applicable in the Arabic characters.

In the Arabic Pascal, the comment is enclosed between two curly brackets. The comment can appear wherever a blank can in the source program. Therefore, a comment is not allowed to be within a single identifier or a number. Whenever the scanner encounters a comment, it simply skips that text and deals with it as if it were a single space.

5.1.1 APC Tokens

The Arabic Pascal has several types of tokens: reserved words, identifiers, strings, special symbols, and numbers.

A) Reserved Words

There are 35 distinct reserved words in the Arabic Pascal language (see Appendix B). Every time a word is encountered, the scanner has to check if this word is a reserved word. For this reason, searching for reserved words should be done efficiently. Therefore, instead of putting all the reserved words in one string array, the Arabic Pascal scanner groups them based on their length to speed up the search process. There are eight groups of reserved words with an average of four to five in each group. Based on the length of the scanned word, only the reserved words with same length are checked. Moreover, the reserved words in each group is alphabetically ordered. As mentioned in Chapter 2, some reserved words can be written in more than one way, such as 'إذا' and 'اذا' which are considered as one reserved word equivalent to 'if'. Therefore, there are a total of 52 reserved words.

B) Identifiers

Any word that is not a reserved word is considered to be an identifier. An identifier can be a program name, a procedure name, a function name, a constant, a type, a variable, or a directive (there is only one directive in the Arabic Pascal "لاحق" which is a translation of the Standard Pascal "FORWARD"). No blanks may appear within an identifier. The APC limits the identifier length to 80. Due to the reasons explained in Chapter 2, the APC accepts the underscore character as part of the identifier.

C) Strings

A string in the Arabic Pascal is enclosed between two double-quote characters and it is not allowed to use a double-quote character inside a string. The reason for using double-quote instead of single-quote (as in the Standard Pascal) is that the Arabic language usually uses double-quote for quotation.

D) Special characters (Symbols)

Special symbols in the Arabic Pascal consist of either one or two characters. Examples of one-character symbols are +, [, <, and :. Examples of two-character symbols are =>, =:, and <>. Note that almost all double-character symbols are written in reverse order with respect to the English Pascal (=: instead of :=) for an obvious reason: the Arabic language reads from right to left. Note also that the greater-than symbol is < and the smaller-than symbol is >. See Appendix B, Section B.4 for the list of symbols in the Arabic Pascal.

E) Numbers

Scanning numbers may be the hardest part to implement in the scanner (see Appendix C for the syntax of the Arabic Pascal numbers). The Arabic Pascal can accept either the Arabic numbers (0,1,2,3,...,9) or the Indian numbers (٠، ١، ٢، ٣، ...، ٩). The character 'س' in the Arabic Pascal is equivalent to the 'E' or 'e' character in the English Pascal. The range of *integer* in the Arabic Pascal is between -32768 and +32767. The scanner limits the number of digits to 20 digits including the signs [BTA93]. It also limits the exponent value in the range of -37 to +37 [BTA93].

Fortunately, the Arabic language and the English language read numbers in the same way: from left to right. That means, both languages consider the left-most digit as the highest significant digit. For this fact, accumulating the value of a number is the same in both languages. It should be pointed out that the comma in the Indian numbers stands for the decimal point. Therefore, under almost all Arabized software, the dot key in the keyboard is printed on the computer screen as a comma in the case of choosing to write in the Indian numbers set.

5.1.2 The Scanner Algorithm

The scanner is called by the parser which requests the next token. Whenever the scanner is invoked, it returns the next token from the source code. The general algorithm for getting a token is as follows:

```
skip-blanks;
select (based on the first character)
    case alphabet: get-identifier;
    case digit:    get-number;
    case quote:    get-string;
    case end-of-file: return end-of-file;
    else           get-symbol;
```

Hence, there are four different algorithms for extracting tokens:

1. **Get-identifier.** It follows syntax diagram (2), Appendix C. The identifier starts with an alphabet character followed by no more than 80 alphabets, digits, or underscores. If the identifier is among the reserved words, the specific reserved word is returned to

the caller (parser), otherwise an identifier token is returned to the parser to insert it in the symbol table.

2. **Get-number.** It follows syntax diagram (29), Appendix C. There are three possible types of numbers: integer, decimal floating-point (e.g., 34.23), and scientific notation which has the e-notation, 'س' in Arabic Pascal, (e.g., 23e4 and 4.5e-5). When the scanner encounters a dot, it has to check that it is not followed by another dot, if so then the double dots are another token used for subranges.
3. **Get-string.** It simply returns the sequence of characters, including spaces, that is enclosed by double-quote character. The new-line character is not acceptable within the string.
4. **Get-symbol.** There are 21 symbols, five of which are double-character symbols: -, +, *, /,), (, =,], [, ^, ، , ؛ , ., .., :, :=, <, <=, <>, >, >=.

When the scanner is invoked for the first time, it opens the source file and prints some header information (such as file-name and time of compilation) in the compilation output.

5.2 THE PARSER

The part of a compiler that knows the source language's syntax is the parser [Hen90]. The parser controls the translation process since it analyzes the source program based on the syntax. It calls the scanner to provide it with tokens. It also handles symbol table functions. Finally it calls the code generator to generate the assembly code.

The APC is designed as a top-down parser (recursive descent) [ASU86, Pys88, PP92, Eld94]. Top-down parsing constructs the parse tree from the root downwards to the terminal symbols at the leaves. In other words, the parser starts at the topmost non-terminal symbol of the grammar down to the terminal ones. The recursive descent parsing method is so called because it is implemented by a set of recursive procedures, one to correspond to each non-terminal of the grammar [Eld94]. The Arabic Pascal parser follows RL(1) grammar: Right-to-left scan, Left-most derivation with one look-ahead token. The RL(1) is exactly as LL(1) for the English language. The "Left-most derivation" of the grammar assumes that the grammar of the Arabic Pascal is written in the conventional way (from left-to-right) with Arabic terminal words and English words for non-terminals. The one look-ahead token approach infers that there is no backtracking. An advantage of following top-down parsing is the ease of construction, and once constructed, it is also easy to read and develop. The main drawback, however, is the slow speed in execution and difficulties in code optimization.

An alternative approach is bottom-up parsing. This parsing technique is harder to implement but has some advantages [Hol90, p.337]. The main advantages are the ability to parallelize the process of parsing and the feasibility of automating the parsing. This approach starts with the terminal symbols and works its way up to the topmost non-terminal symbol. In other words, the tree is constructed from leaves of the parse tree to its root.

5.2.1 Syntax Diagrams

In order for the parser to work correctly, it needs to know the exact description for the source language syntax. There are several approaches to describe a language. Among these approaches are context-free grammar and syntax diagrams [ASU86]. The Arabic Pascal is relatively a simple language that can be fully recognized using syntax diagrams [JW74]. Syntax diagrams are graphical representations of the syntax rules. Appendix C gives a full description of the Arabic Pascal syntax via syntax diagrams. However, not all the details of the language rules can be expressed by the syntax diagrams (or any other approach). For example, syntax diagrams cannot indicate the limits of integer value, and they cannot express the type checking. The Arabic Pascal parser is designed in a way that reflects the syntax diagrams. Of course, the syntax diagrams that are representing the lowest-level syntactic entities, such as identifier and number tokens, are used to write the scanner.

In Appendix C, syntax diagrams 35-39 give a complete syntax description of the Arabic Pascal expressions. They also implement the full operator precedence as follows (See Appendix B, for the Arabic operators):

Operator	**precedence**	
not	1	(highest)
*, /, *div, mod, and*	2	
+, -, or	3	
=, <, >, <>, <=, >=, *in*	4	(lowest)

5.2.2 Symbol Tables

The symbol table is used to maintain all needed information about all identifiers. It must be able to insert, access, and update information efficiently. Therefore, *Binary Search Tree* (BST) is chosen as the data structure for symbol table. In this BST, each identifier and its associated information form a node. Figure 5.2 shows the data structure of the identifier node in the symbol table. The *identifier classification* determines the identifier role in the program such as: label, constant, type, variable, procedure name, etc. Based on the *identifier classification*, certain information is kept in the node. The *Serial number* is used to assign a unique number for each identifier. The use of it will be explained later in Section 5.3. The *scope* is a means to keep track of the identifier scope. The *size* is used to determine the size in bytes of the identifier. This information is utilized by the code generator to generate the required memory space for each identifier. The sizes of different types are presented in Table 5.1. The size of compound types, such as arrays and records, are the total size of their elements. *Left Ptr* and *Right Ptr* are used to construct the BST. In V*ariable* node (Figure 5.2 (b)(8)), if *Relation-Ptr* is *Null*, the variable has no reference to type section such as the variable *t* in the following declaration:

t : (hour, minute, second)

On the other hand, the *Other Information* is omitted if the *Relation-Ptr* is not *Null.* In *Formal parameter* node (Figure 5.2 (b)(9), the *Relation-Ptr* cannot be *Null* because the type of the formal parameter variable is either a simple type or a type that is defined in the type section. The *Next-Ptr* is used in the *Variable* node to relate all variables of the same type that are defined in one statement. For example,

j, k, l : integer;
n, m : integer;

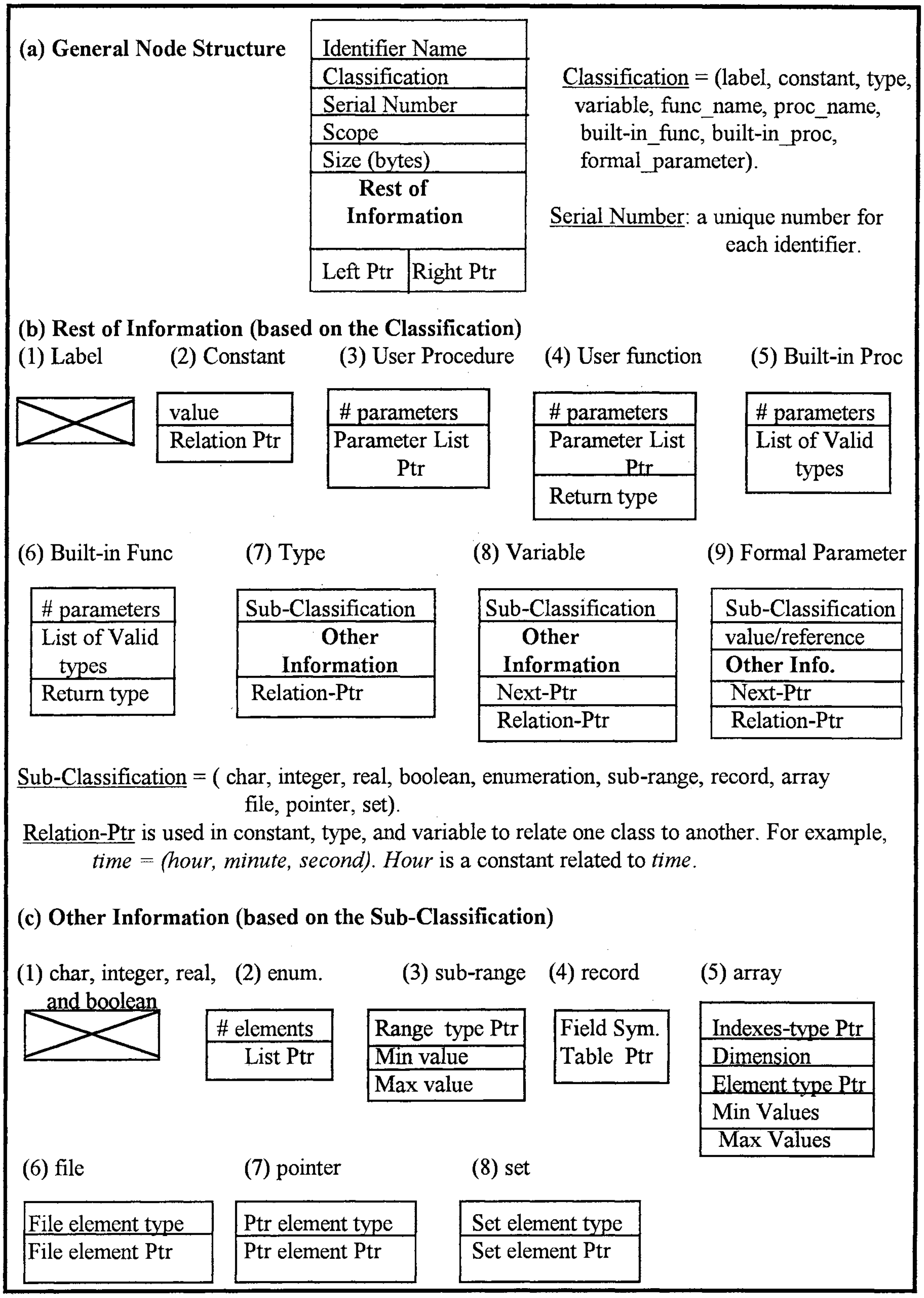
The *Next-Ptr* will link *j, k,* and *l* together, and link n and *m* together, but there is no relation between the two lists except they are of the same type. The *Next-Ptr* is essential because the type definition of variables comes after listing the variables. When the parser encounters the type definition (which is in the previous example "integer") it fills the rest of information based on that type. The *Next-Ptr* in the *Formal Parameter* node is used to relate all formal parameters of one procedure or function.

An example of a symbol table structure is shown in Figure 5.3. For the sake of simplicity, the predefined types' nodes are not shown in the figure. The *int-ptr* and the *char-ptr* are pointers to the integer type node and char type node, respectively.

To reduce the symbol table size, to avoid naming conflict in the symbol table, and to preserve the principle of scope easily, the APC constructs more than one symbol table. The main two symbol tables are that contain the pre-defined identifiers and the user's global identifiers, if any. For each local procedure or function, a separate symbol table is created. Whenever an identifier is referenced in the source code, the search starts from the local scope to the surrounding scopes until it is found or the main symbol table is reached.

The main source of feeding symbol tables is the declaration part of the program, procedure, or function. The global symbol table is initialized with pre-defined types, constants, functions, and procedures.

- The predefined types are: *integer, real, boolean, char,* and *text*. Type *boolean* is an enumeration type with two constant identifiers *true* and *false*.
- The pre-defined constants are: *true, false,* and *nil*.
- The pre-defined procedures are: *read, readln, write, writeln, new, dispose, reset,* and *rewrite*.
- The pre-defined functions are: *abs, arctan, chr, cos, eof, eoln, exp, ln, odd, ord, pred, round, sin, sqr, sqrt, succ,* and *trunc*.



Classification = (label, constant, type, variable, func_name, proc_name, built-in_func, built-in_proc, formal_parameter).

Serial Number: a unique number for each identifier.

Sub-Classification = (char, integer, real, boolean, enumeration, sub-range, record, array file, pointer, set).

Relation-Ptr is used in constant, type, and variable to relate one class to another. For example, *time = (hour, minute, second). Hour* is a constant related to *time*.

Figure 5.2 Symbol Table Node Data Structure

Example

```
CONST
   length = 10;
TYPE
   h = array [1.. length] of  CHAR;
VAR
   a, q : h;
   j, x : INTEGER;
```

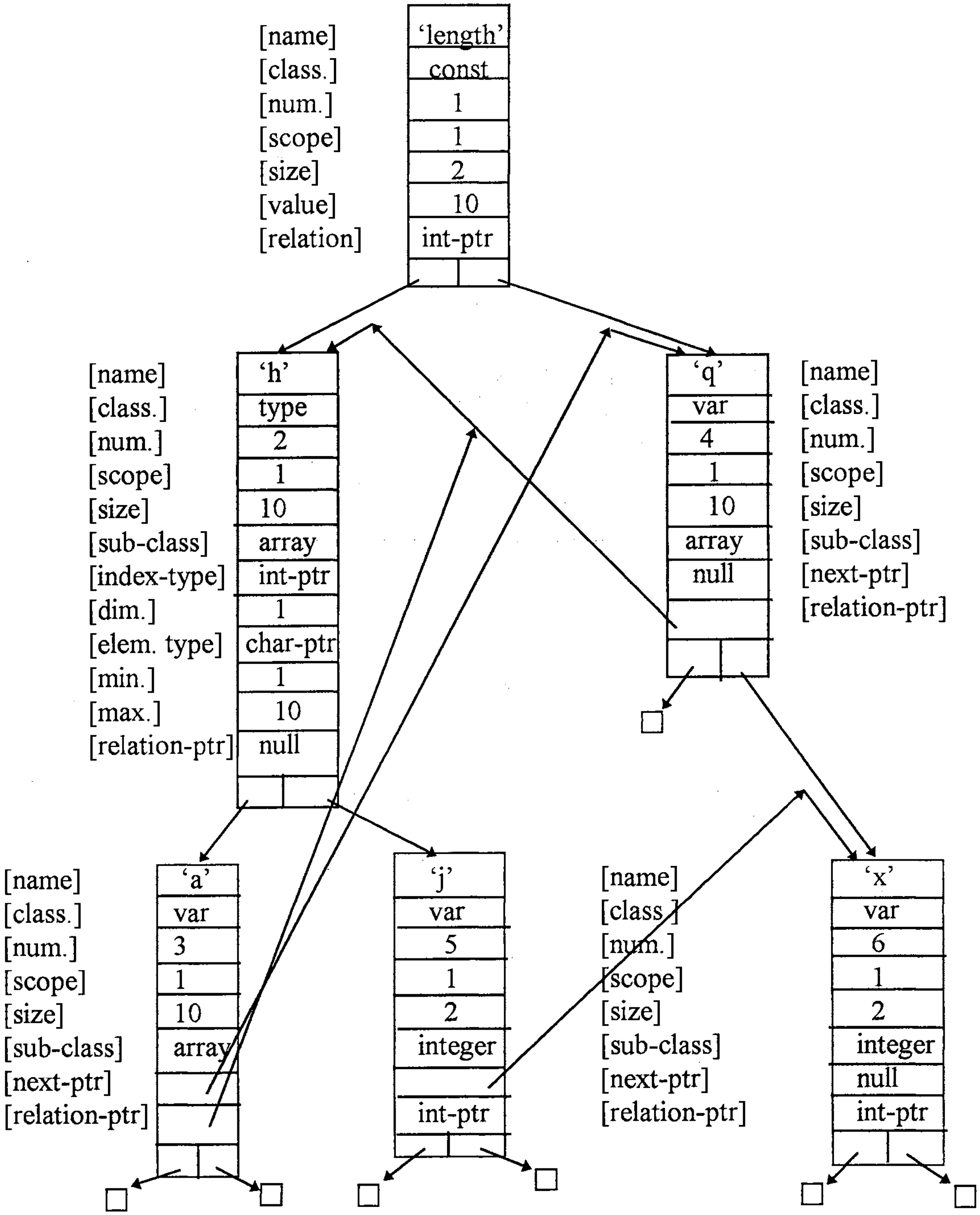


Figure 5.3 An Example of a symbol table

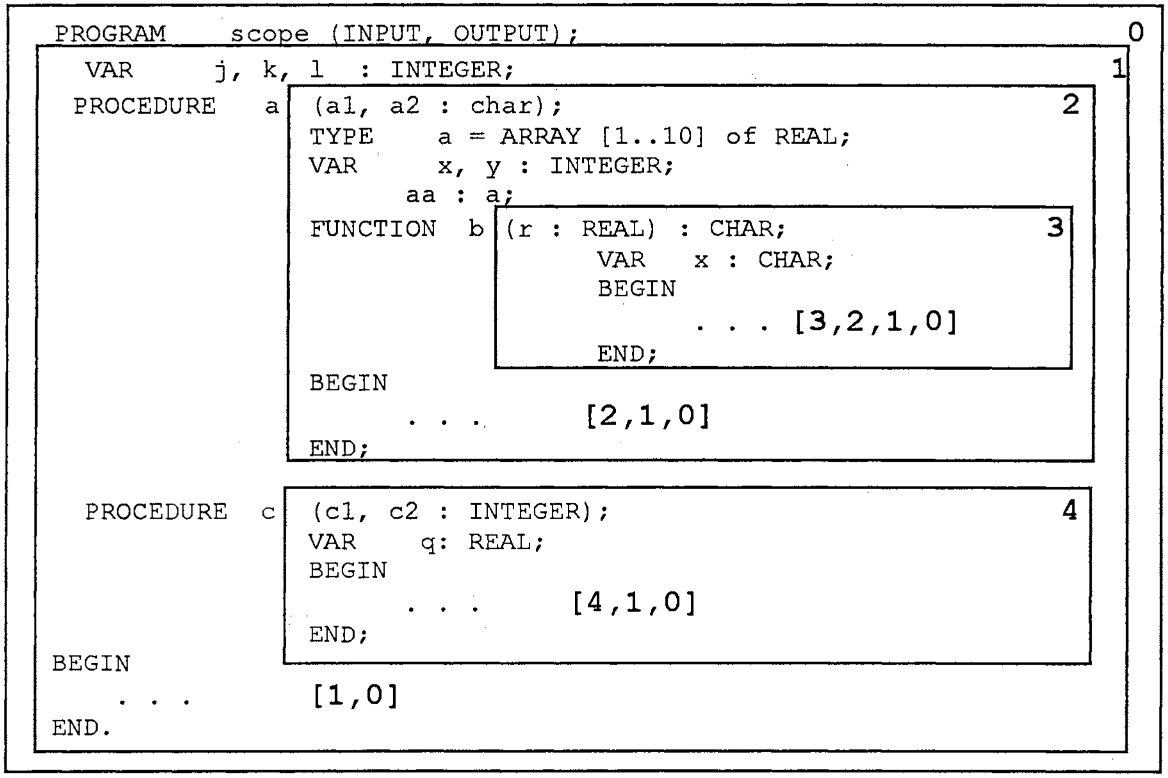
Table 5.1 Sizes (in bytes) of different data types

Type	int.	bool.	real	char	enum.	array	pointer	file	record
Size	2	1	4	1	2	size*type	2	2	Σ of its fields

5.2.3 Identifier's Scope

The scope of an identifier is the portion of the program where that identifier can be referenced [ASU86]. The Arabic Pascal allows the same identifier in a program to name different variables as long as the variables belong to different scopes. The scope rules of the Arabic Pascal determine which declaration of an identifier applies when that identifier appears in the source code. An identifier can be a constant, type, variable, parameter, procedure, or a function. An identifier is said to be *global* if it is a predefined identifier, type, procedure, function, etc., or if it is declared in the declaration section of the main program itself [Hol90]. An identifier is called *local* if it is declared inside a procedure or a function. In the APC, there are only two levels of global identifiers that are defined during the entire compilation process. The first level is the predefined types, identifiers, procedures, and functions. The second level is the global identifiers, if any, that are defined by the programmer in the main program. On the other hand, there are as many levels (scopes) of local identifiers as the number of nested functions and procedures. Figure 5.5 depicts the concept of the identifier scope via an example. Each scope level (block) has been enclosed in a box. The block number is shown in the upper right of the corresponding box. Within the statement part of each block the number of the blocks whose local identifiers are accessible at that point are enclosed in brackets. The lookup for an identifier starts from the local block outward to enclosing block. Note that the numbers in the brackets are sorted according to the look up procedure.

The concept of identifier's scope plays an important role in the parsing process. Whenever the parser encounters an identifier in the body of a procedure or function (including the main program), the parser first checks the *with-stack* (a stack that is associated with *with* statements), if not empty, to concatenate the identifier with the *with-stack* entries, one at a time, starting from the stack top, then it checks the symbol table associated with that procedure or function (local scope). If it did not find the identifier, it checks the next enclosing scope. The search continues outward within enclosing scopes towards the outermost scope (global scope). The search stops either by finding the identifier or reaching the outermost scope. In the latter case, an error message is issued to the programmer.



```
PROGRAM    scope (INPUT, OUTPUT);                                   0
  VAR      j, k, l  : INTEGER;                                      1
  PROCEDURE   a (a1, a2 : char);                               2
                TYPE     a = ARRAY [1..10] of REAL;
                VAR      x, y : INTEGER;
                      aa : a;
                FUNCTION  b (r : REAL) : CHAR;                3
                              VAR   x : CHAR;
                              BEGIN
                                     . . . [3,2,1,0]
                              END;
                BEGIN
                      . . .      [2,1,0]
                END;
  PROCEDURE   c (c1, c2 : INTEGER);                            4
                VAR    q: REAL;
                BEGIN
                      . . .     [4,1,0]
                END;
BEGIN
    . . .        [1,0]
END.
```

Figure 5.5 Scope structure of a Pascal program

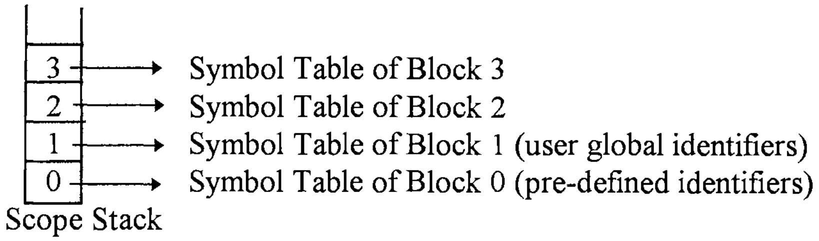

3
2
1
0
Symbol Table of Block 3
Symbol Table of Block 2
Symbol Table of Block 1 (user global identifiers)
Symbol Table of Block 0 (pre-defined identifiers)
Scope Stack

Figure 5.6 A Snapshot of the Scope Stack of the program in Figure 5.5

As stated earlier, each procedure or function (including the main program) is associated with a symbol table containing all the local identifiers. To implement the principle of scope, symbol tables are grouped in a stack, called the *scope stack*. Initially, the parser pushes the pre-defined identifiers' symbol table (in the bottom of the scope stack). Appendix B, Sections B.2 and B.3 list the pre-defined types and identifiers, and the built-in procedures and functions, respectively. The parser, then, pushes the global identifiers in the *scope stack*. When the parser enters a new subprogram (procedure or function), it pushes the subprogram's symbol table in the *scope stack*. As soon as the parser processes a subprogram, its corresponding symbol table is deleted to decrease memory requirements during the compilation process. Therefore, the search for an identifier starts from the top of the stack to its bottom until the identifier is found. An error message is issued for unsuccessful search. Figure 5.6 shows a snapshot of the scope stack of the program in Figure 5.5 while the parser in `function b()`.

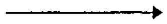
5.2.4 The parser Algorithm

Most of the APC coding belongs to the parser. The Arabic Pascal parser consists of two handlers and five sub-parsers. The handlers are the symbol table handler and the synchronization handler. The sub-parsers are subprogram parser, declaration parser, statement parser, expression parser, and standard procedures and functions parser. All the sub-parsers are implemented according to the structure of the Arabic Pascal syntax diagrams (as in Appendix C).

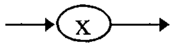
Before explaining the details of the parser, it is important to know the approach of mapping the syntax diagrams into algorithms. Each syntax diagram in the Arabic Pascal can be decomposed into the following simple cases:

1. Primitive case:

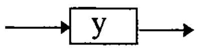
This case has three forms:



by-pass



consume



call

where, the circle represents a terminal symbol (token) and the rectangle represents a non-terminal symbol. The by-pass arrow in a syntax diagram is the arrow that allows by-passing (skipping) one or more syntax entities (terminal or non-terminal symbols). This arrow is used to skip optional structures. The second form makes the parser consume the token (x) and continues parsing. The last form makes the parser call subprogram y.

2. Branching case:

This case has two forms:

(a) Forward branching. The branch is forward if it does not return to a previous state in the syntax diagram. The parser implements this using a *case* (select) statement (if more than two branches) or an *if* ... *else* statement (if only two branches and one of them is a by-pass, then the parser uses an *if* statement).

(b) Backward branching. The branch is backward if it returns to a previous state in the syntax diagram. The parser implements this using either *while* statement (if there is a by-pass arrow that permits skipping the entire backward branch) or *repeat* statement.

3. Merging case:

When two or more arrows in a syntax diagram meet, the parser ends their corresponding branching statements.

Therefore, the first step in implementing the parser is to map all syntax diagrams into simple structures. This scheme handles only the correct structures, so we need to add error checking to the parser. Moreover, we need to embed type checking for all statements and expressions. The final step is to call the code generator to generate the equivalent assembly code where necessary.

The following subsections briefly explain the main functions of each handler or sub-parser. For the sake of simplicity and brevity, we will mention only the main syntax diagram(s) that the sub-parser is following (without writing its (their) corresponding algorithm(s)). In the first sub-parser (Section 5.2.4.3), we will give the corresponding algorithm for syntax diagram (3) as an illustrative example.

5.2.4.1 Symbol Table Handler

As stated earlier, symbol tables in the APC are constructed utilizing the binary search tree concept. They differ form the plain binary search trees in two ways. First, they have some extra pointers that relate all variables of the same type, and relate each variable with its type. Second, in the case of records, the record node is considered as the root for the fields' nodes. In other words, for each record, there is a symbol table associated with it within the main symbol table (nested symbol tables). The symbol table handler carries out three main functions: symbol table initialization, insertion, and search.

(a) Initialization

```
initialize the scope stack's top with zero.
insert all predefined constants, types, functions, and procedures.
          (The scope for all identifiers is set to zero).
```

(b) Insertion

```
search the current symbol table to ensure the uniqueness of the identifier within
    that scope.
```

create a new node and insert all needed information.

insert the node in its appropriate position in the symbol table.

(c) search

There are two types of searches: searching only the current symbol table (mainly used for insertion) and searching the entire scope stack starting from top to bottom (mainly used when referencing an identifier). In the latter case, the *with stack* is checked before starting the search to concatenate its content (as a leading string) to the identifier. Since the *with stack* can only be associated with the local scope, the *with stack* is checked only when searching the current local scope. The *with stack* is also checked from top to bottom. Let us take the following example:

```
. . .
with    rec, ptr^ do              {equivalent to: with rec do  with ptr^ do }
        x := . . .;
        . . .
```

The *with stack* will have two items: *rec* and *ptr^* (*ptr^* in the top). The search goes as follows

1. search for *ptr^.x* in the local symbol table, if not,
2. search for *rec.x* in the local symbol table, if not,
3. search for *x* in the local symbol table, if not,
4. search for *x* in the next symbol tables.

5.2.4.2 Synchronization Handler

When the parser encounters an error, it tries to recover from the error as soon as possible. This step is important because the parser has to continue syntax checking in a meaningful way [Bac79]. The main idea behind error recovery is to skip tokens until reaching a token that a parser can fully recognize. When the parser reaches this point, it re-synchronizes itself at that point and then continues the process of parsing. This type of error recovery is called panic-mode recovery [ASU86]. To implement this idea, the parser has to determine some synchronization points where syntax checking can be reliably restarted (such as semicolon or *begin*). Therefore, when the parser finds a synchronization point, it resumes its normal parsing as if there were no errors. Another source of errors is type mismatch where the compiler expects an identifier of a specific type and it encounters an identifier of another type. Type mismatch does not usually require a synchronization.

On the other hand, error correction is, in general, an unsuccessful approach because it is time consuming with little chance of getting the right solution. Further more, most syntax errors are usually easy to figure out manually, but the problem is with semantic errors which are almost impossible to be automatically corrected and at the same time is hard to be figured out manually.

5.2.4.3 Subprogram Parser

It mainly consists of two parts: main-program parser and procedure/function parser and both of them call a common parser called the body parser.

(a) Main-program parser

```
call the scanner initializer.
call the code generator to generate the main-program prolog.
call the symbol table initializer to initialize the symbol table (create the pre-defined identifier symbol table).
initialize the with-stack to be empty (it is used by the with-statement).
call the program-body parser to parse the entire program. It follows syntax diagram (1). The program-body parser calls body parser which parses syntax diagram (3).
call the code generator to generate the main-program epilog.
```

if not all labels in labels_list, if any, are marked label_seen and goto_seen call error_handler.

generate a compilation summary which includes the number of lines compiled and the number of errors.

(b) Procedure/function parser

parse the header of the subprogram, including the parameter list, if any (there should be no parameter list if the subprogram has been defined before using forward). After parsing the subprogram name, the parser enters a new scope. It follows either syntax diagram (8) or (9) based on whether it is a procedure or a function. For parsing the parameter list it follows syntax diagram (13). If the directive "forward" is encountered restore the previous scope and exit this parser.

call the code generator to generate the appropriate subprogram prolog.

call the body parser to parse the subprogram body (syntax diagram (3))..

call the code generator to generate the appropriate subprogram epilog.

if not all labels in labels_list, if any, are marked label_seen and goto_seen call error_handler.

returns to the previous scope (pops the scope stack).

A *labels_list* is used to keep track of the usage of labels. A *goto* statement and its corresponding label have to be in the same subprogram (including the main program). When a label is encountered, it is inserted in the *labels_list*, if not in, and marked as *label_seen*. When a goto statement is encountered, its label is inserted in the *labels_list*, if not in, and marked as *goto_seen*. At the end of the subprogram, all labels in the *labels_list* should be marked as *label_seen* and *goto_seen*, or at least *label_seen*. For example,

label 10;

```
begin
        . . .
        GOTO        10;
        . . .
end.
```

There is no label in the code for 10 (which is incorrect).

(c) The Body Parser

The algorithm follows syntax diagram (3).

```
if      token = "label"       call    label parser.
if      token = "const"       call    constant parser.
if      token = "type"        call    type parser.
if      token = "var"         call    variable parser.
while  token = "function"  OR  "procedure"
        call  procedure/function parser.
call  statement parser.
```

5.2.4.4 Declaration Parser

It mainly consists of four parts: label parser, constant parser, type parser, and variable parser.

(a) Label parser

The algorithm follows syntax diagram (4). A label must be an unsigned number between 0 and 9999 [Tib81].

(b) Constant parser

The algorithm follows syntax diagram (5). A constant can be a number (with or without a sign), a string, or a constant identifier.

(c) Type parser

The algorithm follows syntax diagram (6).

Other than the predefined types (integer, real, char, boolean, and text), there are seven types: enumeration, subrange, set, file, array, record, and pointer. Each of which can be composition of two or more types within certain rules. In other words, a type can be formed using another type. For example

days = (Sat, Sun, Mon, Tue, Wed, Thu, Fri);

days_set = set of days;

So, *days_set* is a set of another type, enumeration type. The following is a brief description about parsing each type.

1. Subrange type. The lower and the upper values can be of any ordered, simple type (excluding real type). It also accepts the previously defined enumeration data. The lower and the upper values must be from the same type and the lower value is less than or equal the upper value. The algorithm follows syntax diagram (28).

2. Enumeration type. It consists of an ordered sequence of integer constant identifiers which are enclosed between parentheses. The value of the first identifier is zero, the value of the second one is 1, and so on. The algorithm follows syntax diagram (27).

3. Set type. It is a compound data type that accepts only subrange type or enumeration type. The number of the elements in a set should be at most 16 elements. Each element is represented by one bit according to its presence in the list. The algorithm follows syntax diagram (31).

4. File type. It accepts only integer, real, or character types. The file of characters is equivalent to *text* type. The algorithm follows syntax diagram (33).

5. Array type. It involves two other types: indexes' types and elements' type. The array can be of single or multiple dimensions. The parser verifies the validity of each dimension. The element type can be of any type. The size of the array (in bytes) is calculated based on the number of dimensions, the length of each dimension, and the size of the element. The algorithm follows syntax diagram (30).

6. Record type. The fields of the record are kept in a private symbol table that is accessible only through the record. Therefore, the same name of a field can be used outside the record without any ambiguity. For example,

```
rec = record
        j, k : integer;
end;
j, k : real              { or whatever }
```

is a valid definition because the j and k (inside the record) belong only to the record (in a separate symbol table). Therefore, they can be accessed either explicitly by preceding them by the record name, or inside a *with* statement. Other than these two cases, any reference to j or k it is a reference to the outer ones.

The record size is the sum of its fields' sizes. The algorithm follows syntax diagram (32). Note that parsing the record fields is similar to parsing the *var* section.

7. Pointer type. All previous types must be defined before they can be used in the definition of other types, except the pointer type. The pointer type, however, may point to a type that is defined later or to a type that is still being defined. For example:

```
ptr = ^rec;              { 'ptr' is the left side, 'rec' is the right side}
rec = record
        next : ^rec;     { = ptr}
        ....
end;
```

When parsing the definition of *ptr*, the *rec* identifier is not defined yet. When parsing the field *next*, the *rec* is not yet complete. The APC keeps a list of all undefined types (for pointers only). At the end of type parsing, all types should be defined so the parser will be able to resolve all type definitions. The algorithm follows syntax diagram (18).

(d) Variable parser

Parsing this section is almost similar to the type section with three exceptions.

1. A declaration may have one or more identifiers separated by commas, followed by the type specification. All variables that have the same type are linked together.
2. All type specifications should be previously defined including the pointer type.
3. The parser calls the code generator to generate the appropriate memory reservation for all defined identifiers.

The algorithm follows syntax diagram (7).

5.2.4.5 Statement Parser

A statement may start with a label, if so, the parser parses it and calls the code generator to generate a label. The statement parser, then, calls the appropriate subprogram based on the first token in the statement. There are nine possible tokens to start a statement. The algorithm follows syntax diagrams (10) and (14).

1. Identifier. An identifier may stand for either a variable, function or a procedure name.

(a) Variable and function name. The statement has to be an assignment statement. The assignment statement should have the following format:

variable (or function name) := expression;

{ the parameters of the function should be omitted}

The statement parser calls the code generator to generate code for assignment. The expression parsing will be discussed later. The algorithm follows syntax diagram (19).

(b) Procedure name. If the procedure is a standard procedure, then the parser calls *standard- procedures-and-functions* parser. In case of user-defined procedures, the parser checks the actual parameter list, if any, for the correctness of them compared with the formal parameters. It also checks which of these parameters are called by reference and which are called by value. Then, it calls the code generator to generate the appropriate code. The algorithm for user-defined procedure call follows syntax diagram (25).

2. ‘ابدأ’ **(begin).** This means a compound statement. So, the parser calls the statement parser until it reaches the corresponding *end*. The algorithm follows the last part of syntax diagram (3).

3. ‘اذهب_إلى’ **(goto).** The parser calls the code generator to generate a jump instruction. The parser checks for the presence of the target label in the code within the same scope. The algorithm follows syntax diagram (26).

4. ‘إذا’ **(if).** The parser calls the expression parser to evaluate the boolean expression, then it again calls the statement parser. If ‘وإلا’ (else) is encountered, another call for statement parser is issued. The algorithm follows syntax diagram (20).

5. ‘كرر’ **(repeat).** The parser calls the statement parser until the token is ‘حتى’ (until). It, then, calls the expression parser to evaluate the expression. The algorithm follows syntax diagram (23).

6. ‘طالما’ **(while).** The parser calls the expression parser to evaluate the boolean expression, then it again calls the statement parser. The algorithm follows syntax diagram (22).

7. ‘لقيم’ **(for).** The parser parses the header of the *for* statement checking the control identifier, its type, its lower value, and its upper value. It also calls the code generator to generate the appropriate assembly code. It, then, calls the statement parser to parse the body of the *for* statement. The algorithm follows syntax diagram (24).

8. ‘حالة’ **(case).** The parser parses the header of the *case* statement by calling the expression parser. For the *case* values, the parser calls the code generator to generate assembly code as if it were nested *if ... else* statements (see Section 5.3.1). Since the *case* statement in the Standard Pascal does not have a default value, the inner most *if* does not have an *else*. The algorithm follows syntax diagram (21).

9. 'مع' (with). *With* argument(s) should be of record type. The parser pushes the argument(s) into the *with-stack* (see Section 5.2.3). The statement parser, then, parses the enclosed statement(s). *With* statement is the only statement that has an effect on the structure of the identifiers. For example, instead of writing the full identifier name, such as *rec.ident*, only the *ident* may appear inside the *with* statement. Section 5.2.4.1 part (c) discussed how the search is performed in the symbol table for an identifier. The algorithm follows syntax diagram (40).

5.2.4.6 Expression Parser

The expression parser is the part of the parser that is responsible for parsing expressions, evaluating them, and ensuring the correctness of their usage. It is invoked to parse

- the actual parameters of procedures and function,
- the boolean expressions in the control statements, and
- the expressions after the assignment statements.

The expression parser calls the code generator to generate the appropriate code.

All expressions (including variables and constants) preserve their actual values when evaluated except set variables and set constants. As stated earlier, each element in a set is represented by one bit according to its position in the list. Therefore, sets need special handling in evaluating their expressions. For example, let us consider the following code (in Standard Pascal):

```
(1)     s1 : set of 5 .. 10;
(2)     s2:  set of 1 .. 10
```

```
       . . .
(3)    s1 := [5, 7, 9];
(4)    s2 := [5, 7, 9];
(5)    if (6 in s1) then . . .
(6)    if (6 in s2) then . . .
       . . .
```

In statement (3) the numbers 5, 7, and 9 mean nothing to set ***s1*** except that they represent the fist, the third, and the fifth bits, respectively. While in statement (4) the same numbers represent fifth, seventh, and ninth bits, respectively, for ***s2***. The value of ***s1*** after executing statement (3) should be 21 (i.e., in binary 0000000000010101) and the value of ***s2*** after executing statement (4) is 336 (i.e., in binary 0000000101010000). Similarly, number 6 in statement (5) and (6) only represents the second bit with respect to ***s1,*** and the sixth bit with respect to ***s2***. Consequently, when the expression parser encounters a statement that involves a set expression, it performs the following:

1. if the set elements are of subrange type, calculate the element-value by subtracting the value of the variable or constant from the minimum value in the set declaration. The minimum value of a enumeration type is always zero. This gives the bit position in the set.
2. to set or test a particular bit in the set, a run-time function is called to find the power of 2 of the element-value.
3. the assignment statement (such as statement (3) above) is considered as a series of unions of individual set elements.

   Using this approach of implementation makes a restriction on using an immediate set comparison in the Arabic Pascal. For example,

```
          if [5, x*3]  <= s1  then . . .          {where x is an integer}
```

is not acceptable in the Arabic Pascal because the value of the set *[5, x*3]* is unknown unless this set is associated with a specific set definition. To implement the previous code, the programmer needs to add an assignment statement as follows:

```
        s3 := [5, x*3];                { assuming s3 of the same s1 type}
        if s3 <= s1  then . . .
```

In this way, the compiler knows exactly the value of the set *[5, x*3]*.

The *expression parser* algorithm follows syntax diagrams (35) and (37).

Note that the Arabic Pascal *function* has two forms. If it is in the left hand side of an assignment statement (usually inside the function body itself), it does not take any parameter. On the other hand, if it is inside an expression (such as *x * sin (x)*), then it is parsed as if it were a procedure call.

5.2.4.7 Standard Procedures-and-Functions parser

It is not difficult to parse the standard functions and procedures that have a fixed number of arguments. The parser checks the correctness of the number of arguments and their types, and then calls the code generator to generate the appropriate code. Table 5.2 shows the standard functions, the acceptable types of parameters, and the type of the result. Note that all standard functions have only one parameter except *eof* and *eoln* which may have zero or one parameter. Table 5.3 shows the standard procedures, number of parameters, and the acceptable types for each parameter. Note that procedures GET and PUT are not implemented in the Arabic Pascal because they are functionally equivalent to simple read and write respectively. The procedure PAGE is also not implemented because it was used for non-interactive computers to order the printer to jump to a new page [Got94, p.77]. Many modern versions of Pascal eliminated these functions, such as Turbo Pascal and VAX Pascal.

The problem arises when parsing the procedures that have variable number of arguments, namely اكتب، اكتب_س، اقرأ، اقرأ_س (*write, writeln, read,* and *readln).* The easiest way is to parse each of which as if it were a sequence of procedures that having only one parameter. Therefore, after parsing each parameter, the parser calls the code generator to generate the appropriate code. The actual parameters of اقرأ و اقرأ_س *(read* and *readln)* must be variable. The actual parameters of اكتب و اكتب_س (*write* and *writeln*) may be expressions and each of which may be followed by field width and designators.

Table 5.2 Standard functions

Function	type of the parameter	type of the result
abs and sqr	integer or real	same as the parameter type
arctan, cos, exp, ln, sin, and sqrt	integer or real	real
chr	integer	char
odd	integer	boolean
ord	char	integer
pred and succ	integer, real, or boolean	same as the parameter type
round and trunc	real	integer
eof and eoln	file (if exist)	boolean

Table 5.3 Standard procedures

Procedure	# of parameters	types of parameters
dispose and new	1	pointer
reset and rewrite	2	1. file 2. string
read	1 or more (if the first is not file) 2 or more (if the first is file)	1. file, integer, real, or char (variable) 2 .. n integer, real, or char (variable)
readln	0 or more (if the first is not file) 1 or more (if the first is a file)	1. file, integer, real, or char (variable) 2 .. n integer, real, or char (variable)
write	1 or more (if the first is not file) 2 or more (if the first is a file)	1. file, integer, real, char, boolean (expression), or string 2 .. n integer, real, char, boolean (expression), or string
writeln	0 or more (if the first is not file) 1 or more (if the first is a file)	1. file, integer, real, char, boolean (expression), or string 2 .. n integer, real, char, boolean (expression), or string

All math functions follow syntax diagram (45).

Functions *eof* and *eoln* follow syntax diagram (46).

Procedures *new* and *dispose* follow syntax diagram (47).

Procedures *reset* and *rewrite* follow syntax diagram (48).

Procedures *read* and *readln* follow syntax diagrams (49) and (50).

Procedures *write* and *writeln* follow syntax diagrams (51), (52), and (53).

5.2.5 Type Checking

One of the major tasks of the parser is to check the validity of using the identifiers and the compatibility among related identifiers. This task is called *type checking*. The type checking relies heavily on the symbol tables to perform its task. There are four common categories of type checking:

1. **Variables:** A variable can be a simple identifier, an array, a record, a pointer, a set, or a file. Each of the above has a specific way of usage. For example, a subscript is usually associated with array variables. The type checker checks the identifier's definition in the symbol table to ensure correctness of usage. Type checking of an identifier also depends on the syntax surrounding that identifier. For example, a pointer identifier that is followed by a pointer symbol has a different semantic from a plain pointer identifier.
2. **Expressions**: The type checker has to check whether or not the operands of an operator are of the correct type. For example, both operands of AND operator must be of type *boolean*. In expressions, the type checker must specify the overall expression result type. For instance, an integer plus a real results in a real expression. Moreover, all arithmetic relations should be between two similar types (except the IN set relation where the first operand has to be from the same members' type of the second one). The relations less-than and greater-than are not defined in sets. It is also not acceptable to check the relation between two strings unless they have the same length[Tib81].

3. **Assignments**: In the Arabic Pascal, one type is assignment compatible with another type if an expression of the first type can be assigned to a variable of the second type. Type checker checks whether or not an expression value of one type can be assigned to a variable of a different type. For example, a real variable can get an integer value but not vice versa.
4. **Statements**: The type checker must ensure that all variables and expressions in the statement are of the correct types. For example, all control statements should have expressions of *boolean* type. As another example, in the *for* statement, both the initial-value and the final-value have to be from the same type.

5.3 THE CODE GENERATOR

The final step in the compiling process is code generation. Whenever the parser encounters a complete statement or expression, it calls the code generator to generate an equivalent assembly code. The APC translates the source code into 8086 assembly language. The details of the 8086 structure and assembly language can be found in [MP86, Abe87, Hah87, BB88, BTA93, Mis93].

For the sake of simplicity, all standard procedures and functions are implemented as a run-time library. The APC calls this library whenever needed. This run-time library must be linked with the generated assembly code. It should be pointed out that the APC allocates global variables in the data segment instead of at the bottom of the stack to allow direct access to them.

The code generator generates at the beginning of each assembly program a set of directives called a *prolog* [BTA93]. These directives give the Turbo Assembler the needed information about the code. Similarly, at the end of the assembly code, some restoring actions are done and a list of global variables is placed, called *epilog* [BTA93].

The Turbo Assembler seems not to accept the numbers within the Arabic identifiers nor the Arabic underscore, so the code generator should eliminate these from the identifiers. This will have no effect on the uniqueness of the identifiers within the same scope because the identifier serial number will make every identifier unique in the entire assembly program. The code generator attaches a unique number (the serial number) to every identifier in the program. The need for the uniqueness of identifiers is necessary because the 8086 assembly language does not have the scope concept.

Another issue in the code generation is the conditional jump. The conditional jump instructions in the original 8086 assembly language are restricted to a distance of 127 bytes [BTA93]. On the other hand, the unconditional jump can jump to any place within the code segment. Therefore, all conditional jumps that may cause out of range jumps should be converted somehow to equivalent unconditional jumps. For example,

```
        cmp    x, y          ; compare
        je     exit          ; jump if equal to exit label

        .

        . <may be more than 127 bytes of code>

        .

exit:   <the rest>
```

This code should be altered to avoid the restriction. The code generator converts it to the following:

```
        mov    temp, 1       ; move one to temp
        cmp    x, y
        je     next1
        xor    temp, temp    ; make temp = 0
next1:  cmp    temp, 1
        je     next2
        jmp    exit          ; unconditional jump
next2:  .
        . <may be more than 127 bytes of code>
        .
exit:   <the rest>
```

The code generator works in two levels: statement level and expression level. The following subsections discuss these two levels.

5.3.1 Statement Code Generator

There are eleven different types of statements that the code generator may face:

1. assignment statement. The code generator generates code for evaluating the right hand side expression (stored in AX or AX:DX for real values), then issues a move instruction from AX to the left hand side variable.

2. goto statement. It simply issues a jump instruction.

3. compound statement. It does nothing.

4. with statement. It does nothing.

5. procedure/function call. It pushes all the arguments (or their addresses for reference arguments) on the stack, then it calls the function or the procedure. In the case of function, the return value is stored in register AX. In the procedure or function itself, it saves the stack pointer at the beginning and restores it at the end.

6. standard procedure/function call. It pushes all the arguments on the stack, then it calls the equivalent function in the run-time library. In the case of function, the result is stored in register AX.

7 . **while statement.** The general format for the *while* statement is

while <expression> *do* <statement(s)>

The code generator generates the following code:

```
    loop_begin:

                <code for expression>

                mov    Reg, 1         ; move one to Reg

                jXX    next           ; XX is the conditional jump, such as jle

                xor    Reg, Reg             ; let Reg equal 0

    next:       cmp    Reg, 1               ; compare

                je     statement_label      ; jump if equal

                jmp    loop_exit            ; unconditional jump

    statement_label:   <code for statement(s)>

                jmp    loop_begin

    loop_exit:
```

8. repeat statement. The general format for the *repeat* statement is

repeat <statement(s)> until <expression>

The code generator generates the following code:

```
    loop_begin:

                <code for statement(s)>

                <code for expression>

                mov    Reg, 1         ; move one to Reg

                jXX    next           ; XX is the conditional jump, such as jle
```

```
            xor    Reg, Reg          ; let Reg equal 0
next:       cmp    Reg, 1            ; compare
            je     loop_exit         ; jump if equal
            jmp    loop_begin        ; unconditional jump
loop_exit:
```

9. **if statement.** The general format for the *if* statement is

```
        if    <expression>        then    <statement(s)-1>
    or
        if  <expression>  then  <statement(s)-1>  else   <statement(s)-2>
```

The code generator generates the following code:

```
            <code for expression>
            mov    Reg, 1            ; move one to Reg
            jXX    next              ; XX is the conditional jump, such as jle
            xor    Reg, Reg               ; let Reg equal 0
next:       cmp    Reg, 1                 ; compare
            je     then_label             ; jump if equal
            jmp    if_exit                ; unconditional jump (no else)
      OR    jmp    else_label             ; unconditional jump (with else)
then_label:
            <code for statement(s)-1>
            jmp    if_exit                ; (with else)
else_label:        <code for statement(s)-2>    ; (with else)
if_exit:
```

10. **for statement.** The general format for the *for* statement is

```
    for    id := initial_val     [to/downto]   final_val  do  <statement(s)>
```

The code generator generates the following code:

```
                <code for intial_val>
                mov    current_val, initial_val
for_loop:       <code for final_val>
                cmp    current_val, final_val
                jle    continue                       ; in case of ‘to’
           OR   jge    continue                       ; in case of ‘downto’
                jmp    loop_exit
continue:       <code for statement(s)>
                inc    current_val          ; add one in case of ‘to’
           OR   dec    current_val          ; subtract one in case of ‘downto’
                jmp    for_loop
loop_exit:
```

11. case statement. The general format for the *case* statement is

```
        case    <expression> of
          <const1-1> [ , <const1-2>, . . . , <const1-n] :     <statement(s)-1>;
          <const2-1> [ , <const2-2>, . . . , <const2-m]:     <statement(s)-2>;
                . . . . .
        end;
```

The code generator generates the following code:

```
                <code for expression>          ; the result is stored in register AX
                cmp    AX, <const1-1>
                je     branch_1
                cmp    AX, <const1-2>
                je     branch_1
                .....
                cmp     AX, <const1-n>
                je      branch_1
                jmp     next_1
branch_1:       <code for statement(s)-1>
                jmp     case_exit
next_1:         cmp     AX, <const2-1>
                je      branch_2
                cmp     AX, <const2-2>
                je      branch_2
                .....
                cmp     AX, <const2-m>
                je      branch_2
                jmp     next_2
branch_2:       <code for statement(s)-2>
                jmp     case_exit
next_2:         ....
case_exit:
```

5.3.2 Expression Code Generator

Most of the relations and operators in the Arabic Pascal has a direct equivalent instruction in the assembly code. Table 5.4 shows the simple relations and operators that have one to one correspondence between the Arabic Pascal and the 8086 assembly language. Of course, this does not include the preparation of the operands nor the details of type conversions.

Table 5.4 Simple relations and operations in Arabic Pascal and Assembly

Arabic Pascal	8086 Assembly	Arabic Pascal	8086 Assembly
=	je	- (not sets)	sub
<>	jne	* (integer)	imul
=> (not sets)	jle	* (sets intersection)	and
<= (not sets)	jge	قاسم (div)	idiv
>	jl	:=	mov
<	jg	و (and)	and
+ (not sets)	add	أو (or)	or
+ (sets union)	or	ليس (not)	not
- (unary)	neg	في (in)	test

The rest of relations and operations may not have simple equivalence.

1. Real operations. Since handling real numbers are not directly supported by the 8086 assembler, the APC calls a run-time library to accomplish the real operations [Hah87]. Note that the division (/) always works with real operands (the integer operands are automatically converted to real in the APC).

2. 'باقي' (remainder). When making a *idiv* instruction in the 8086 assembly, the quotient is placed in the register AX and the remainder is placed in DX [Hah87]. So, the code generator performs a *idiv* instruction but gets the DX result. Note that some Pascal references [BTP89, JW74, Tib81] describe the *mod* operator with the same behavior as the remainder operator.

3. Set difference (-). In sets, the difference between two sets (s1 - s2) is equivalent to (s1 and (not s2)). Therefore, the second operand should be negated and, then, *anded* with the first one.

4. Set inclusion (=>, in Arabic Pascal). In sets, ($s1 \subseteq s2$) is equivalent to ($s1 - s2 = \varnothing$). Therefore, the inclusion is implemented using the difference and set equality.

5. Set containment (=<, in Arabic Pascal). In sets, ($s1 \supseteq s2$) is equivalent to ($s2 - s1 = \varnothing$). Therefore, the containment is also implemented using the difference and set equality.

5.4 THE ERROR HANDLER

Because programmers are prone to making syntax errors, it is necessary for the compiler to handle these errors accurately and efficiently. When the APC encounters an error, it simply does the following:

1. Gives the programmer the nearest possible location of the error by pointing to it. The error handler prints the line number that has the error and the current token. Usually the error happens at the current token or in a previous token in the same line. For example, (in Standard Pascal):

   if (I < J)　then　I := J; else　J := I;

   The APC cannot discover that the semicolon before *else* is an error until it parses the *else* token.

2. Prints out a descriptive error message.

Appendix D, Section D.1 lists the possible error messages the programmer may get from the compiler during program development.

The run-time errors are generated by the run-time library that takes the control when abnormal situation takes place during the execution. Appendix D, Section D.2 lists the possible run-time errors.

5.5 EXAMPLES OF ARABIC PASCAL PROGRAMS

The following (next pages) are two simple Arabic Pascal programs with their output. For convenience, their corresponding English Pascal programs are shown in the same lines. Remember that the Arabic language reads from right to left. More Arabic Pascal programs can be found in Appendix E.

Example 1

```
{ The Towers of Hanoi, Using
  recursive procedure calls  }

PROGRAM  towerofhanoi(INPUT, OUTPUT);

VAR
     n : INTEGER;       { Number of disks  }

  PROCEDURE  transfer(n, origin, dest, other :
INTEGER);

     { Transfer n disks from the original
       to the destination  }

     PROCEDURE diskmove(origin, dest :
INTEGER);
        { Move a single disk from the original
          to the destination }
     BEGIN
          WRITELN('Move  ', origin:1, '  to
', dest:1);
     END;   { diskmove }
  BEGIN  { transfer }
         IF (n > 0)  THEN
         BEGIN
              transfer(n-1, origin, other,
dest);
              diskmove(origin, dest);
              transfer(n-1, other, dest,
origin)
         END
  END;  {  transfer  }

BEGIN    { main  }
         WRITE('Enter the number of disks: ');
         READLN(n);
         WRITELN;
         transfer(n, 1, 3, 2)
END.
```

```
{ برنامج أبراج هانوي، وذلك باستخدام عملية
تكرارية  }

برنامج  برج_هانوي(مدخل، مخرج)؛

متغير
  ع : صحيح؛          { عدد الحلقات  }

 عملية  نقل(ع، البداية، النهاية، الوسيط
: صحيح)؛

   { نقل عدد ع من الحلقات من البداية
إلى النهاية  }

   عملية تحريك_الحلقة (البداية، النهاية
: صحيح)؛
   { نقل حلقة واحدة من البداية إلى
النهاية }
     ابدأ
        اكتب سطر("حرك حلقة من "،
البداية:١، "  إلى "، النهاية:١)؛
     نهاية؛   { تحريك_حلقة }
  ابدأ  { نقل  }
        إذا (ع > ٠)  فإن
        ابدأ
             نقل(ع-١، البداية، الوسيط،
النهاية)؛
             تحريك_الحلقة (البداية،
النهاية)؛
             نقل(ع-١، الوسيط، النهاية،
البداية)
        نهاية
  نهاية؛  { نقل  }

ابدأ    { البرنامج الأساسي }
        اكتب("أدخل عدد الحلقات : ")؛
        اقرأ_سطر(ع)؛
        اكتب سطر؛
        نقل(ع، ١، ٣، ٢)
نهاية.
```

Sample Output

```
Enter the number of disks: 4
Move  1  to  2
Move  1  to  3
Move  2  to  3
Move  1  to  2
Move  3  to  1
Move  3  to  2
Move  1  to  2
Move  1  to  3
Move  2  to  3
Move  2  to  1
Move  3  to  1
Move  2  to  3
Move  1  to  2
Move  1  to  3
Move  2  to  3
```

عينة من النتائج

```
أدخل عدد الحلقات : ٤
حرك حلقة من ١ إلى ٢
حرك حلقة من ١ إلى ٣
حرك حلقة من ٢ إلى ٣
حرك حلقة من ١ إلى ٢
حرك حلقة من ٣ إلى ١
حرك حلقة من ٣ إلى ٢
حرك حلقة من ١ إلى ٢
حرك حلقة من ١ إلى ٣
حرك حلقة من ٢ إلى ٣
حرك حلقة من ٢ إلى ١
حرك حلقة من ٣ إلى ١
حرك حلقة من ٢ إلى ٣
حرك حلقة من ١ إلى ٢
حرك حلقة من ١ إلى ٣
حرك حلقة من ٢ إلى ٣
```

Example 2

```
{ Sorting N elements using a bubble sort  }
PROGRAM  sorter(INPUT, OUTPUT);
TYPE
    vector = array [1..50] of INTEGER;
VAR
    a : vector;
    i, size : INTEGER;

  PROCEDURE  sort (VAR list:vector;
n:INTEGER);
    { Sort the first N elements of 'lsit' }
  VAR
     j, temp : INTEGER;
     noexchange : BOOLEAN;
  BEGIN
    REPEAT
       noexchange := TRUE;
       FOR j := 1 TO n-1  DO
         IF list[j] > list[j+1] THEN
           BEGIN
                temp := list[j];
                list[j] := list[j+1];
                list[j+1] := temp;
                noexchange := FALSE;
           end
    UNTIL noexchange;
  END;  { sort }

BEGIN    { main program }
    WRITE('How many values are to be sorted?
');
    READLN(size);
    WRITELN('Enter data : ');
    FOR i := 1 to size  DO
        READ(a[i]);
    sort(a, size);
    WRITELN;
    WRITELN('The sorted Data :  ');
    FOR i := 1 to size  DO
        WRITE(a[i]:5);
END.
```

```
{ ترتيب عدد ن من العناصر باستخدام طريقة
الفقاعة }
برنامج المرتب(مدخل، مخرج)؛
نوع
 قائمة = مصفوفة [١..٥٠] من صحيح؛
متغير
 ق : قائمة؛
 ل، طول: صحيح؛

 عملية ترتيب (متغير أرقام : قائمة؛
ن:صحيح)؛
  { ترتيب أول ن من "أرقام" }
 متغير
  ك، مؤقت : صحيح؛
  بلاتغيير : بولي؛
 ابدأ
  كرر
   بلاتغيير := صواب؛
   بداية_من ك:= ١ إلى ن-١ اعمل
    إذا أرقام[ك] > أرقام[ك+١] فإن
     ابدأ
      مؤقت := أرقام[ك]؛
      أرقام[ك] := أرقام[ك+١]؛
      أرقام[ك+١] := مؤقت؛
      بلاتغيير := خطأ ؛
     نهاية
  حتى بلاتغيير؛
 نهاية؛ { ترتيب }

ابدأ  { البرنامج الرئيسي }
 اكتب("كم عدد الأرقام التي تريد
ترتيبها ؟ ")؛
 اقرأ_سطر(طول)؛
 اكتب_سطر("أدخل الأعداد : ")؛
 بداية_من ل := ١ إلى طول اعمل
  اقرأ(ق[ل])؛
 ترتيب(ق، طول)؛
 اكتب_سطر؛
 اكتب_سطر("الأرقام مرتبة تصاعديا : ")؛
 بداية_من ل:= ١ إلى طول اعمل
  اكتب(ق[ل]:٥)؛
نهاية.
```

Sample output

```
How many values are to be sorted? 5
Enter data :
 4  1  9  7  3

The sorted Data :
    1    3    4    7    9
```

عينة من النتائج

```
كم عدد الأرقام التي تريد ترتيبها ؟ ٥
أدخل الأعداد :
٤  ٩  ١  ٧  ٣

الأرقام مرتبة تصاعديا :
١    ٣    ٤    ٧    ٩
```

CHAPTER 6

CONCLUSION AND FUTURE WORK

The objective of this project is to build an Arabic programming language environment. The first step towards achieving such objective is to Arabize the environment that enables programmers to write programs in Arabic easily and efficiently. The environment is analogous to the programming environments that are available such as Turbo Pascal and Borland C. Therefore, the environment enables programmers to edit, compile, and execute programs. The dissertation describes, in detail, the APE design and implementation

The second step towards achieving the objective is to Arabize the Standard Pascal by designing an Arabic Pascal Compiler. The need for such an Arabized programming language is great in many Arab countries, especially in the education field. Currently available compilers are developed for languages whose lexical units are defined using the English alphabet. For students whose native language is not English, this approach imposes a burden. Therefore, it will be helpful to develop compilers with lexical units in other languages. This dissertation describes the specifications, design, and implementation of a Pascal programming language and its compiler with lexical units defined in the Arabic language. The Arabic Pascal language is an Arabized version of the Standard English Pascal. I have chosen the Standard Pascal as the target language because of its popularity.

The analysis, design, and implementation of the scanner, the parser, the code generator, and the error handler were fully carried out in this dissertation.

Section 6.1 specifies the hardware and software requirements to run this software. Section 6.2 states the limitations of this work. Finally, Section 6.3 discusses possible future work.

6.1 SYSTEM REQUIREMENTS

The system requirements are as follows.

- An IBM PC (or a compatible computer).
- An Arabic version of the MS-Windows (the APE has been tested on MS-Windows version 3.1 and version 3.11).
- At least 4MB of RAM, to be able to run the Arabic Version of MS-Windows and, then, the APE.
- Executable files of the *Turbo Assembler* and the *Turbo Linker*. The APC compiles the source programs to assembly language. The assembly code, then, has to be assembled using the *Turbo Assembler* and has to be linked via the *Turbo Linker*. The APC has been tested using *Turbo Assembler* versions 3.0 and 4.0, and *Turbo Linker* versions 3.0, 5.1, and 6.0. However, Any version after 3.0 of both of them should work. The APC produces an assembly code with the small memory model. The *Turbo Linker* requires *c0ws.obj* (the standard MS-Windows initialization file), *cws.lib* (the small model runtime library for MS-Windows), *mathws.lib* (the MS-Windows math library for small model), and *import.lib* (the library that provides access to the built-in Windows functions). All these files should be distributed with the software.
- About 1MB of hard drive. The size of the executable files of APE, APC, and their complementary files is about 900K without any compression.

6.2 SYSTEM LIMITATIONS

Any system has some limitations. The main limitations of this software are as follows.

1. The APE does not guarantee a full Arabic interface. There are many reasons for interacting with English. Among these are the file names and system critical errors. The file name is recommended to be in English for compatibility reasons, as explained in Chapter 4. The system critical errors are the errors generated by the MS-Windows when an usual event happens, such as memory insufficient or accessing a protected memory area. The Arabic MS-Windows still displays these fatal errors in English. Thus, the APE users may receive some English messages from the system while using the APE.
2. To the best of our knowledge, the APE and APC are free of errors. However, it is not guaranteed that the software is error-free. The exhaustive testing requires several months or even years for such software.
3. The maximum file size is 32K. This is not a large size file, but it is enough for the majority of programs that may be written in this programming language.
4. Even though, the result size can be of any size, the APE cannot send to the printer a result of a size more than 32K, because the APE holds the results up to 32K, then discard the rest.

6.3 FUTURE WORK

There are many open avenues to enhance the current project. Among these avenues are the following.

- Extending the Arabic Pascal by providing more built-in procedures and functions rather than the standard ones, and by utilizing some good programming styles, such as adding an *else* statement to the *case* statement.
- Providing more environment tools such as a runtime debugger within the APE.
- Enhancing the APE editing facilities by introducing syntax directed editing and program auto formatting (indenting).
- Supplying this software with a full documented user's manual in Arabic.
- Expanding the on-line help to provide a full explanation about the APE and the Arabic Pascal language.
- Speeding up the execution time of user programs by generating optimal object codes utilizing the code optimization techniques.
- Overcoming the 32K limitation by increasing the maximum file size (using Visual C++, instead of Visual BASIC) or by allowing more than one file to form a single program (multiple-file program).
- Modifying this software to be available under Windows 95 (currently, there is no Arabic interface in Windows 95).
- Finally, the Arabization of programming languages should be extended to all new concepts of programming languages such as object programming languages and visual programming. The limitation of man-power and the time, force us to choose a simple example, such as Standard Pascal. Such project is the seed for more sophisticated Arabic programming languages.

REFERENCES

[AA88a] Al-Salman, A. and Al-Fantookh, A., "Towards a First Arabization of the UNIX System," *B. Sc. Thesis*, Computer Science Dept., King Saud University, Saudi Arabia, 1988.

[AA88b] AlFedaghi, S. and Amin, A., "Automatic Spelling Correction in Arabic," *Technical Report*, Electrical and Computer Engineering Department, Kuwait University, 1988.

[AA91] Amin, A. and Al-Fedaghi, S., "Machine Recognition of Printed Arabic Text Utilizing Natural Language Morphology," *International Journal of Man-Machine Studies*, Vol. 35, Dec. 1991, pp. 769-789.

[AA93] As-Sfran, S. and Aref, M. M., "التمثيل الدلالي للجمل العربية", *Proceedings of the Arabic Language and Advance Information Technologies: اللغة العربية والتقنيات المعلوماتية المتقدمة*, Casablanca, Morocco, Dec. 8-9 1993, pp. 45-56. In Arabic.

[AA94] Al-Daimi, K.J. and Abdel-Amir, M.A., "The Syntactic Analysis of Arabic by Machine," *Computers and Humanities*, Vol. 28, No. 1, 1994, pp. 29-37.

[Abd90] AbdulKader, A. A., "الحاسوب ودوره في الفهرسة اللفظية للحديث النبوي الشريف," *Proceedings of Using Computers in the Islamic Sciences: ندوة استخدام الحاسوب في العلوم الشرعية*, Jeddah, Saudi Arabia, Nov. 11-13 1990, pp. 77-104. In Arabic.

[Abe87] Abel, Peter, *IBM PC Assembly Language and Programming*, Prentice-Hall, 1987.

[Abu92] Abu-Salem, H.O., "A Microcomputer-Based Arabic Bibliographic Information Retrieval System with Relational Thesaurus," *Ph. D. Dissertation*, Computer Science, Illinois Institute of Technology, USA, 1992.

[AFC94] Arabi, M., Fischthal, S.M., Cheng, V.C., and Bart, E., "Algorithms for Arabic Name Transliteration," *IBM Journal of Research and Development*, Vol. 38, No. 2, March 1994, pp. 183-193.

[AG90] Abo-ElHamayl, M. A. and Gado, S. H., "تعريب نظام دوبيس/ليبس," *Proceedings of the 12th National Computer Conference and Exhibition*, King Saud University, Riyadh, Saudi Arabia, Oct. 21-24, 1990, pp. 498-513. In Arabic.

[AG95] Alimi, A. M. and Ghorbel, O. A., “Error Analysis in an On-Line Recognition System of Arabic Handwritten Characters,” *Proceedings of the 14th National Computer Conference*, Riyadh, Saudi Arabia, 1-4 Apr., 1995, pp. 407-417.

[AH88] Al-Hawaj, A. Y. and Hamed, M., “Design of an Expert System for Teaching Programming Fundamentals,” *Proceedings of the 10th National Computer Conference*, King Abdulaziz University, Jeddah, 28 Feb.-2 March 1988, pp. 835-846.

[AHD94] Abuhaiba, I. S., Holt, M. J., and Datta, S., “Straight Line Approximation and 1D Representation of Off-Line Handwritten Text,” *Image and Vision Computing*, Vol. 12, No. 10, Dec. 1994, pp. 649-659.

[Ahm94] Ahmed, A., “الحاسوب في خدمة السنة النبوية المطهرة,” *Proceedings of the 2nd Computer Arabization Symposium*, King Saud University, Saudi Arabia, 27-30 March 1994, pp. 221-254. In Arabic.

[AK88] Ahmed, J. I. and Khayat, M. G., “Design and Implementation of a Database System for Display, Storage, and Retrieval of the Holy Qur’an,” *Proceedings of the 10th National Computer Conference*, King Abdulaziz University, Jeddah, 28 Feb.-2 March 1988, pp. 774-776.

[Ala90] Alamri, A., “مجال استفادة العلوم الإسلامية من الكمبيوتر,” *Proceedings of Using Computers in the Islamic Sciences: ندوة استخدام الحاسوب في العلوم الشرعية*, Jeddah, Saudi Arabia, Nov. 11-13 1990, pp. 43-52. In Arabic.

[Ald94] Aldarab, I. A., “تطبيقات الحاسب الآلي في العلوم الشرعية,” *Proceedings of the 2nd Computer Arabization Symposium*, King Saud University, Saudi Arabia, 27-30 March 1994, pp. 255-277. In Arabic.

[Ale94] Alemam, O., “التعرف على الكلمات المنطوقة: نظام للغة العربية,” *Proceedings of the 2nd Computer Arabization Symposium*, King Saud University, Saudi Arabia, 27-30 March 1994, pp. 39-46. In Arabic.

[Ali87] Ali, N., “Morphological Storage and Retrieval of Holy Qura’n,” *Proceedings of the First K.S.U. Symposium on Computer Arabization*, King Saud University, Saudi Arabia, April 6-9 1987, p. 78.

[Ali88] Ali, N., “Computer Arabization: A Futuristic View,” *Technical Report*, Alalamiah Hi-Tech, Kuwait, 1988.

[Ali89] Ali, N., “Formalization and Computation of Arabic Syntax,” *Proceedings of the 11th National Computer Conference*, Dahran, Saudi Arabia, 4-7 March 1989. pp. 309-320.

[Ali90] Ali, N., “نظم المعلومات-المشكلات والحلول,” *Proceedings of Using Computers in the Islamic Sciences: ندوة استخدام الحاسوب في العلوم الشرعية*, Jeddah, Saudi Arabia, Nov. 11-13 1990, pp. 117-144. In Arabic.

[Alk90] AlKharashi, I., “An efficient Contextual Analysis Algorithm for Arabic Text Handling,” *Proceedings of the 12th National Computer Conference and Exhibition*, King Saud University, Riyadh, Saudi Arabia, Oct. 21-24, 1990, pp. 465-474.

[Alk91] AlKharashi, I., “Micro-AIRS: A Microcomputer Based Arabic Information Retrieval System , Comparing Words, Stems, Roots, as Index Terms,” *Ph.D. Dissertation*, Computer Science Dept., Illinois Institute of Technology, Chicago, USA, 1991.

[Alk92] Al-Khrisat, M., “Structuring the Arabic Lexicon and Thesaurus with Lexical Semantic Relations to Support Information Retrieval, *Ph.D. Dissertation*, Computer Science Dept., Illinois Institute of Technology, Chicago, USA, 1992.

[Alkh] *Al-Khawarizmi User Manual*, National Computer group. No Year.

[Alm90] Almofti, B. A., “استخدام تقنيات متقدمة في خدمة الشريعة الإسلامية,” *Proceedings of Using Computers in the Islamic Sciences: ندوة استخدام الحاسوب في العلوم الشرعية*, Jeddah, Saudi Arabia, Nov. 11-13 1990, pp. 105-116. In Arabic.

[Almua] Almuarib “المعرّب”, Version 2, King Abdulaziz City for Sciences and Technology. No date. In Arabic.

[Als92] Al-Safran, S.A., “An Arabic Sentence Generator,” *M.S. Thesis*, Computer Science, King Fahad University of Petroleum and Minerals, Saudi Arabia, 1992.

[Als96] Al-Salman, A., “An Arabic Programming Environment,” Proceedings of the 1996 ACM Symposium on Applied Computing, Philadelphia, PA, Feb. 18-20, 1996, pp. 480-486.

[Aly89] Al-Yousefi, H., “Recognition of Handwritten Arabic Characters,” *Ph. D. Dissertation*, Computer Science, Colorado State University, USA, 1989.

[AM87] Ali, N. and Morcos, I., “Data Compression Technique for Arabic Text,” *Proceedings of the First K.S.U. Symposium on Computer Arabization*, King Saud University, Saudi Arabia, April 6-9 1987, pp. 67-77.

[AM95] AlSalman, H. A. and Mandorah, M. M., “استخلاص المعرفة لنظم تعليم الإسلام بمساعدة الحاسوب,” *Proceedings of the 14th National Computer Conference*, Riyadh, Saudi Arabia, 1-4 Apr., 1995, pp. 70-84. In Arabic.

[AM94] Abo-Assamh, H. A. and Manther, A., “محرر نصوص عربي متكيف مع المعاقين محدودي الحركة,” *Proceedings of the 2nd Computer Arabization Symposium*, King Saud University, Saudi Arabia, 27-30 March 1994, pp. 119-126. In Arabic.

[AMA94a] Al-Bawab, Marwan, Mrayati, Muhammad, Alam, Y. M., and Tayyan, M. H., “A Computerized Morpho-Syntactic System of Arabic,” *The Arabian Journal for Science and Engineering*, Vol. 19, No. 3, July 1994, pp. 461-480.

[Ama94b] Amarah, Jamal, *Microsoft Visual Basic*, Vol. 1 & 2, AlMiman, 1994. In Arabic.

[AMG94] Abuhaiba, I.S., Mahmoud, S.A., and Green, R.J., "Recognition of Handwritten Cursive Arabic Characters," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, Vol. 16, No. 6, June 1994, pp. 664-672.

[Ami88] Amin, A. M., "OCR of Arabic Texts," *Proceedings of the 4th International Conference on Pattern Recognition*, London, Springer, 1988.

[ASM82] ASMO, "ASMO Standard 449," *Arab Organization for Standardization and Metrology*, Amman, Jordan, 1982.

[ASU86] Aho, Alfred V., Sethi, Ravi, and Ullman, Jeffrey D., *Compilers Principles, Techniques, And Tools*, Addison-Wesley, 1986.

[AU92] Al-Yousefi, H. and Udpa, S.S., "Recognition of Arabic Characters," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, Vol. 14, No. 8, Aug. 1992, pp. 853-857.

[AYE90] Al-Sadoun, H., Yaseen M., El-Jallad, A., and El-Jallad M., "ARbic BasIc (ARBI): A New Arabic MS-DOS Based Programming Language," *Proceedings of the 12th National Computer Conference and Exhibition*, King Saud University, Riyadh, Saudi Arabia, Oct. 21-24, 1990, pp. 449-464.

[Bac79] Backhouse, Roland C., *Syntax of Programming Languages*, Prentice-Hall, 1979.

[Bak86] Bakhit, B. B., "Arabic Data Compression, a Case Study,", *" Proceedings of the 9th National Computer Conference and Exhibition*, Riyadh, Saudi Arabia, 1986, pp. 10-4-1 to 10-4-12.

[Bak87] Bakry, S. H., "Towards a Standard Arabic Glossary of Computer Terms," *Proceedings of the First K.S.U. Symposium on Computer Arabization*, King Saud University, Saudi Arabia, April 6-9 1987, pp. 28-36.

[Bas95] *BASIC صخر User Manual*, Al-Alamiah Est. 1995.

[BB88] Brumm, Penn and Brumm, Don, *80386 Assembly Language*, TAB Professional and Reference Books, 1988.

[Bec87] Becker, J.D., "Arabic Word Processing," *Communications of the ACM*, Vol. 30, No. 7, July 1987, pp. 600-610.

[Ber78] Berry, R. E., "Experience with the Pascal P-Compiler, ", *Software-Practice and Experience*, Vol. 8, No. 5, 1978, pp. 617-627.

[BKN86] Booth, L., Khalid, M., Niaz, M., and Al-Waidan, H., "Arabization of an Automated Library," ," *Proceedings of the 9th National Computer Conference and Exhibition*, Riyadh, Saudi Arabia, 1986, pp. 10-3-1 to 10-3-40.

[BTA93] *Borland Turbo Assembler User's Guide*, Borland International, 1993.

[BTP89] *Borland Turbo Pascal User's Guide and Reference Guide*, Borland International, 1989.

[Cas85] Casey, D., "An Original Approach to Arabic," *Middle East Computing*, No. 21, Apr. 1985, p. 31.

[Cha87] Chaher, Y., "Production System of Morphological and Syntactical Analyzers Application to the Arabic Language," *Proceedings of the First K.S.U. Symposium on Computer Arabization*, King Saud University, Saudi Arabia, April 6-9 1987, pp. 45-54.

[CJ88] Capon, P. C. and Jinks, P. J., *Compiler Engineering Using Pascal*, Macmillan, 1988.

[Coo83] Cooper, Doug, *Standard Pascal User Reference Manual*, Norton, 1983.

[Cro94] Crouse, K., *The Windows Programming Puzzle Book*, John Wiley & Sons, 1994.

[DAM92] Dehash, K. A., AlBesher, A. A., Mathkour, H. I., "التخطيط لإدخال الحاسب الآلي في مناهج ومقررات تعليم البنات," *Proceedings of the 13th National Computer Conference*, Riyadh, Kingdom of Saudi Arabia, 28 Nov. - 2 Dec., 1992, pp. 1-29. In Arabic.

[Dav86] Davidson, C., "Arabizing the Micro," *Middle East Computing*, No. 33, May 1986, p. 54.

[Del85] Delwachi, A., "Computer Processing of the Arabic Language: Problem Areas in the Treatment of Arabic in Hardware and Software Systems," *Proceedings of Computer Processing of the Arabic Language*, Vol. 2, Kuwait, April 14-16 1985, pp. 1-26.

[DM86] Dehlawi, F. M. and Madorah, M.M., "ل.ب.أ. : لغة برمجة أخرى," *Proceedings of the 9th National Computer Conference and Exhibition*, Riyadh, Saudi Arabia, 1986, Vol. 2, Section 2, pp. 20-60. In Arabic.

[DP80] Daniels, M. C. and Pemberton, S., "Implementing a Pascal Compiler on an 8085A System," *Microcomputer Applications*, Vol. 4, No. 3, 1980, pp. 144-150.

[Dun86] Duncan, Ray, *Advanced MS-DOS*, Microsoft Press, 1986.

[EB90] El-Imam, Y.A. and Banat, K., "Text-to-Speech Conversion on a Personal Computer," *IEEE Micro*, Vol. 10, Aug. 1990, pp. 62-74.

[EEN89] El-Dessouki, A., El-Dessouki, O., Nazif, A., Ahmed, M., " An ATN Approach for Understanding Arabic Sentences," *The 11th National Computer Conference and Exhibition*, Dahran, Saudi Arabia, 4-7 March 1989, pp. 762-773.

[EG88] El-Sheikh, T. and Guindi, R., "Computer Recognition of Arabic Scripts," *Pattern Recognition*, Vol. 21, No. 4, 1988, pp. 293-302.

[EH89] El-Sadany, T..A. and Hashish, M.A., "An Arabic Morphological System," *IBM Systems Journal*, Vol. 28, No. 4, 1989, pp. 600-612.

[Ela86] El-Affendi, M. A., "نحو لغة متطورة للبرمجة بالعربية ," *Proceedings of the 9th National Computer Conference and Exhibition*, Vol. 2, Section 1, Riyadh, Saudi Arabia, 1986, pp. 14-28. In Arabic.

[Ela87] El-Affendi, M. A., "Efficient Algorithms for Basic System Arabization," *Proceedings of the First K.S.U. Symposium on Computer Arabization*, King Saud University, Saudi Arabia, April 6-9 1987, pp. 37-44.

[Ela90] El-Affendi, M. A., "A Natural Arabic Interface to MS-DOS," *Proceedings of the 12th National Computer Conference and Exhibition*, King Saud University, Riyadh, Saudi Arabia, Oct. 21-24, 1990, pp. 475-497.

[Eld87] El-Dessouki, O., "Dictionary-Based Arabic Text Compression Technique," *Proceedings of the First K.S.U. Symposium on Computer Arabization*, King Saud University, Saudi Arabia, April 6-9 1987, pp. 79-87.

[Eld94] Elder, John, *Compiler Construction: A Recursive Descent Model*, Prentice-Hall, 1994.

[ELS93] El-Hannach, M., Labed, L., Salhi, R., and Ghazali, S., "برنامج لساني–حاسوبي للتعرف على التعابير المسكوكة في اللغة العربية," *Proceedings of the Arabic Language and Advance Information Technologies: اللغة العربية والتقنيات المعلوماتية المتقدمة*, Casablanca, Morocco, Dec. 8-9 1993, pp. 71-90. In Arabic.

[ES89] El-Wakil, M. and Shoukry, A., "On-Line Recognition of Handwritten Isolated Arabic Characters," *Pattern Recognition*, Vol. 22, No. 2, 1989, pp. 97-105.

[Far89] Farghaly, A., "A natural Language Understanding System for Arabic," *Proceedings of the First Kuwait Computer Conference*, Kuwait, March 1989, pp. 595-622.

[Far90] Faraj, M. S. "استخدام الحاسوب الآلي في بناء الموسوعات الشرعية", *Proceedings of Using Computers in the Islamic Sciences: ندوة استخدام الحاسوب في العلوم الشرعية*, Jeddah, Saudi Arabia, Nov. 11-13 1990, pp. 53-76. In Arabic.

[FE95] Fayek, M. and Elhaweet, W., "Computer as an aid for Education," *Proceedings of the 14Th National Computer Conference*, Riyadh, Kingdom of Saudi Arabia, 1-4 Apr. 1995, pp. 85-100.

[Fed92] Feddag, A., "Arabic Morpho-Syntax and Semantic Parsing," *Proceedings of the 13th National Computer Conference*, Riyadh, Saudi Arabia, Nov. 11-14, 1992, Vol. 2, pp. 717-749.

[FF90] Foxley, E. and Feddag, A., " A Syntactic and Morphological Analyzer of Arabic Words," *Proceedings of the 2nd Cambridge Conference on Bilingual Computing in Arabic and English*, Cambridge University, UK, Sept. 5-7 1990.

[FF93] Feddag, A. and Foxley, E., "A Lexical Analyzer for Arabic, " *International Journal of Man-Machine Studies*, Vol. 38, Feb. 1993, pp. 313-330.

[GFA95] Gembi, K. M., Farrash, A. A., and AlHarbi, A. S., "دراسة ميدانية عن تدريس مواد الحاسوب في المرحلة الثانوية," *Proceedings of the 14Th National Computer Conference*, Riyadh, Kingdom of Saudi Arabia, 1-4 Apr. 1995, pp. 2-18. In Arabic.

[Gne94] Gnemi, M. A., "الحاسوب والتعليم," Proceedings of the 2nd Computer Arabization Symposium, King Saud University, Saudi Arabia, 27-30 March, pp. 127-152. In Arabic.

[Gor90] Goraine, H., "Machine Recognition of Arabic Text," *Ph.D. Dissertation*, Computer Science, University of Reading, UK, 1990.

[Gri71] Gries, David, *Compiler Construction for Digital Computers*, John Wiley, 1971.

[GUA92] Goraine, H., Usher, M., and Al-Emami, S., " Off-Line Arabic Character Recognition," *Computer*, Vol. 27, No. 7, July 1992, pp. 71-74.

[Hah87] Hahn, Harley, *The Complete Guide to IBM PC AT Assembly Language*, Scott, Foresman and Company, 1987.

[Ham94a] Hamado, A. "مراحل التحليل الآلي للغة العربية قصد تحسين الأخطاء وتصحيحها," *Proceedings of the 2nd Computer Arabization Symposium*, King Saud University, Saudi Arabia, 27-30 March 1994, pp. 25-38. In Arabic.

[Ham94b] Hammouri, A., "An Arabic Lexical Database to Support Natural Language Processing, *Ph.D. Dissertation*, Computer Science Dept., Illinois Institute of Technology, USA, 1994.

[Han85] Hansen, P. B., *Brinch Hansen On Pascal Compilers*, Prentice-Hall, 1985.

[HE86] Hegazi, N. H. and Elsharkawi, A. A., "Natural Arabic Language Processing," *Proceedings of the 9th National Computer Conference and Exhibition*, Riyadh, Saudi Arabia, 1986, pp. 10-5-1 to 10-5-17.

[Hed75] Hedrick, G. E., Editor, *Proceedings of the 1975 International Conference on ALGOL 68, Section 1- Survey of ALGOL 68 Compilers*, Oklahoma State University, Stillwater, June 10-12, 19975.

[Hen90] Hendrix, J. E., *A Small C Compiler*, Second Edition, M&T Books, 1990.

[Hil90] Hilal, Y., "الحاسوب في خدمة الحديث النبوي الشريف," *Proceedings of Using Computers in the Islamic Sciences: ندوة استخدام الحاسوب في العلوم الشرعية*, Jeddah, Saudi Arabia, Nov. 11-13 1990, pp. 235-253. In Arabic.

[Hol90] Holub, Allen I., *Compiler Design in C*, Prentice Hall, 1990.

[Hol91] Holzner, Steven, *Advanced Assembly Language*, Brady, 1991.

[Ibr91] Ibrahim-Shaker, M.M., "A Fast and Expert Machine Translation System Involving Arabic Language," *Ph. D. Dissertation*, Cranfield Institute of Technology, UK, 1991.

[Hun85] Hunter, R., *Compilers: Their Design and Construction Using Pascal*, John Wiley, 1985.

[IDF89] Ibrahim, A., Douglas, J., and Fahhmy, A., "Arabic in Machine Translation," *Proceedings of the First Cambridge Conference on Bilingual Computing in Arabic and English*, Cambridge University, UK, 1989.

[Jam91] Jambi, K.M., "Design and Implementation of a System for Recognizing Arabic Handwritten Words with Learning Ability," *Ph.D. Dissertation*, Computer Science, Illinois Institute of Technology, USA, 1991.

[JW74] Jensen, K. and Wirth, N., *PASCAL User Manual and Report*, Second Edition, Springer-Verlag, 1974.

[Kal85] Kaldirm, O., "Architecture of Arabic Computer," *Proceedings of Computer Processing of the Arabic Language*, Vol. 1, Kuwait, April 14-16 1985, pp. 1-7.

[KC91] Khan, E.H. and Chaudhry, F.I., "Contextual Analysis Approach for Arabization of a microcomputer," *IEEE Transactions on Consumer Electronics*, Vol. 37, No. 1, Feb. 1991, pp. 86-95.

[KH93] Khayat, M. G. and Hanadi, M. A., "تمثيل الدلالة الصرفية لأوزان الأفعال في النظم الآلية لفهم اللغة العربية," *Proceedings of the Arabic Language and Advance Information Technologies: اللغة العربية والتقنيات المعلوماتية المتقدمة*, Casablanca, Morocco, Dec. 8-9 1993, pp. 57-70. In Arabic.

[Kha86] Khayat, M.G., "Printing Arabic Text Using Dot Matrix Printers," *Software-Practice and Experience*, Vol. 16, No. 2, Feb. 1986, pp. 165-172.

[MA85] Mandurah, M. M. and Al-Musa, A. O., “The use of graphics to generate high quality Arabic characters, ” *Proceedings of the 8th National Computer Conference*, Al-Kobar, Saudi Arabia, Oct. 1-7 1985, pp. 1031-1041.

[MA94] Mandorah, M. M. and AlSalman, H. A., “الحاسوب في خدمة الدعوة الإسلامية: مشروع تطوير نظام لتعليم القرآن لغـير المسـلمين,” *Proceedings of the 2nd Computer Arabization Symposium*, King Saud University, Saudi Arabia, 27-30 March 1994, pp. 197-210. In Arabic.

[Mac90] Mackay, Pierre (Editor), *Computers and the Arabic Language*, Hemisphere Publishing Co., 1990.

[Mah85] Mahjoub, A., “A New Definition of True Arabization,” *Middle East Computing*, No. 27, Nov. 1985, pp. 61-65.

[Mah94] Mahmoud, S., “Arabic Character Recognition Using Fourier Descriptors and Character Contour Encoding,” *Pattern Recognition*, Vol. 27, No. 6, 1994, pp. 815-824.

[Mak91] Mak, Ronald, *Writing Compilers & Interpreters*, John Wiley, 1991.

[Man85] Mandorah, M..M., “Architecture of an Arabic Computer,” *Proceedings of Computer Processing of the Arabic Language*, Vol. 2, Kuwait, April 14-16 1985, pp. 8-26.

[Man90] Mandorah, M. M., “الحاسبات في خدمة الدعوة الإسلامية,” *Proceedings of Using Computers in the Islamic Sciences: ندوة استخدام الحاسوب في العلوم الشرعية*, Jeddah, Saudi Arabia, Nov. 11-13 1990, pp. 191-208. In Arabic.

[Man95] Mandorah, M. M., “أسس ومتطلبـات التخطيـط لمنـاهج التعليـم وبرامـج التدريـب في عصـر المعلومـات, ” *Proceedings of the 14Th National Computer Conference*, Riyadh, Kingdom of Saudi Arabia, 1-4 Apr. 1995, pp. 476-489. In Arabic.

[Mic88] MicroSoft, *MS DOS User's Guide Arabic Supplement*, MicroSoft, 1988.

[Mic92] MicroSoft, *MS-DOS Arabic Support User's Guide*, MicroSoft, 1992.

[Mic93] MicroSoft, *Microsoft Word- دليل المستخدم باللغة العربية*, MicroSoft, 1993. In Arabic.

[Min87] *Project of introducing of computing to public school system*, Ministry of Education, Internal memo, Technical committee, Jan. 87.

[Mis93] Mischel, Jim, *Macro Magic with Turbo Assembler*, John Wiley, 1993.

[MM87] Mahjoub, A. H. and Mandurah, M. M., “Current Issues and Future Directions in Computer Arabization, ” *Proceedings of the First K.S.U. Symposium on Computer Arabization*, King Saud University, Saudi Arabia, April 6-9 1987, pp. 1-27.

[MM94] Mahmood, S. A. and Mandorah, M. M., "بيئة البرمجة كاتب وتطبيقاتها في تطوير نظم تعليم حاسوبية ذكيـة," *Proceedings of the 2nd Computer Arabization Symposium*, King Saud University, Saudi Arabia, 27-30 March 1994, pp. 47-94. In Arabic.

[MM95] Mankai, Chafia and Mili, Ali, "Machine Translation from Arabic to English and French," *Information Sciences*, Vol. 3, No. 2, March 1995, pp. 91-109.

[Mor90] Morfeq, A.H., "BAYAN: A Text Database Management System for Arabic Engineering Documents," *Ph.D. Dissertation*, Computer Science, University of Colorado, USA, 1990.

[Mou87] Mouajed, M. B., "APG INGRES: An Arabic Interface for INGRES Base," *Proceedings of the First K.S.U. Symposium on Computer Arabization*, King Saud University, Saudi Arabia, April 6-9 1987, pp. 127-134.

[MP86] Murray, William H. and Pappas, Chris H., *80386/80286 Assembly Language Programming*, Osborne McGraw-Hill, 1986.

[MSF88] Mosich, D., Shammas, N., and Flamig, B., *Advanced Turbo C Programmer's Guide*, John Wiley, 1988.

[MT95] Mostafa, S. M. and Takrony, Y. S., "Computer Assisted English to Arabic Translation," *Proceedings of the 14th National Computer Conference*, Riyadh, Saudi Arabia, 1-4 Apr., 1995, pp. 429-442.

[NA94] Noah, A. S. and Ahmed, A., "التعرف على الحروف العربية: استعراض دقيق ," *Proceedings of the 2nd Computer Arabization Symposium*, King Saud University, Saudi Arabia, 27-30 March 1994, pp. 1-24. In Arabic.

[Najla] *Najla User Manual*, Saudi Computer Industries. No year.

[NEE88] Nazif, A., El-Dessouki, O. I., El-Dessouki, A., and Ahmed, M., "An Expert System for Understanding Arabic Sentences," *Proceedings of the 10th National Computer Conference*, King Abdulaziz University, Jeddah, 28 Feb.-2 March 1988, pp. 745-760.

[NH94] Narayanan, A. and Hashem, L., "Finite-State Abstractions on Arabic morphology,", *Artificial Intelligence Review*, Vol. 7, No. 6, 1994, pp. 373-399.

[NM91] Narayanan, A. and Mehdi, S., "A Computer Model for Transliterated Arabic," *Applied Computer Translation*, Vol. 1, No. 3, 1991, pp. 5-28.

[PD82] Pemberton, S. and Daniels, M. C., *Pascal Implementation: The P4 Compiler*, John Wiley, 1982.

[Pem80] Pemberton, S., "Comments on an Error-Recovery Scheme by Hartmann," *Software-Practice and Experience*, Vol. 10, No. 3, pp. 231-240.

[PP92] Pittman, Thomas and Peters, James, *The Art of Compiler Design*, Prentice-Hall, 1992.

[Pys88] Pyster, Arthur B., *Compiler Design and Construction*, Van Nostrand Reinhold, 1988

[QHA93] Qafari, A. S., Hamruni, B. M., Al-Kharashi I. A., and Evens, M. W., “خطوات تطبيقية نحو منهجية مدعمة بالحاسب الآلي لمعالجة ونشر المصطلـح العربي,” *Proceedings of the Arabic Language and Advance Information Technologies: اللغـة العربيـة والتقنيـات المعلوماتيـة المتقدمـة*, Casablanca, Morocco, Dec. 8-9 1993, pp. 259-278. In Arabic.

[RS93] Rafea, A.A. and Shaalan, K.F., “Lexical Analysis of Inflected Arabic Words Using Exhaustive Search of an Augmented Transition Network,” *Software-Practice and Experience*, Vol. 23, No. 6, June 1993, pp. 567-588.

[Sam72] Sammet, J. E., “Programming Languages: History and Future,” *Communications of the ACM*, Vol. 15, No. 7, 1972, pp. 601-610.

[Sau86] Saudi Soft., *Al Mussaed Al Arabi*, Jeddah, Saudi Arabia, 1986.

[SC92] Suleiman, Khalid A. and Citrin, Wayne, “An International Visual Language,” *Proceedings. of IEEE Workshop on Visual Languages*, Seattle, Sept. 1992.

[SC95] Suleiman, Khalid A. and Citrin, Wayne, “A Visual Approach to Programming Language Internationalization: An Arabic Case Study,” to appear *in Language Problems and Language Planning*, Vol. 19, No. 2, 1995, pp. 113-132.

[Sch89] Schildt, Herbert, *Advanced Turbo C*, Second Edition, McGraw Hill, 1989.

[Smi90] Smith, B., “Around the World in Text Displays,” *Byte*, Vol. 15, No. 5, May 1990, pp. 262-268.

[TA90] Tayli, M. and Al-Salamah, A.I., “Building Bilingual microcomputer Systems,” *Communications of the ACM*, Vol. 33, No. 5, May 1990, pp. 495-504.

[Tab93] Tabaza, K., “Windows on Arabia,” *Byte*, Vol. 18, No. 7, July 1993, p. 48.

[Tay87] Tayli, M., “Integrated Arabic System,” *Proceedings of the First K.S.U. Symposium on Computer Arabization*, King Saud University, Saudi Arabia, April 6-9 1987, pp. 135-143.

[Tek90] Tekfi, C., “Design of a Computer Information System for the Algerian National Archives,” *Ph.D. Dissertation*, Information Science, The City University, London, UK, 1990.

[Tib81] Tiberghien, Jacques, *The Pascal Handbook*, Sybex Computer Books, 1981.

[TNS86] Tayli, M., Nafisah, M., and Shahin, S., “Intelligent Arabic Workstation,” *Proceedings of the 9th National Computer Conference and Exhibition*, Riyadh, Saudi Arabia, 1986, pp. 10-2-1 to 10-2-8.

[UN84] United Nations Development Program Regional Bureau for Arab States, "*Computer, Informatics, and Development in Some Arabic-Speaking Countries*," Mission Report, N.Y., July 1984.

[WH86] Welsh, J. and Hay, A., *A Model Implementation of Standard Pascal*, Prentice-Hall, 1986.

[Wir71] Wirth, N., "The Design of a Pascal Compiler," *Software-Practice and Experience*, Vol. 1, No. 4, 1971, pp. 309-333.

[WQ72] Welsh, J. and Quinn, C., "A Pascal Compiler for the ICL1900 Series Computers," Software-Practice and Experience, Vol. 2, No. 1, 1972, pp. 73-77.

[You88] Young, Michael, *MS-DOS Advanced Programming*, Sybex, 1988.

[ZRN87] Zyoute, M., Rajouani, A., Najim, M., Ouadou, M., and Chiadmi, "Speech Synthesis of the Arabic Language," *Proceedings of the First K.S.U. Symposium on Computer Arabization*, King Saud University, Saudi Arabia, April 6-9 1987, pp. 88-89.

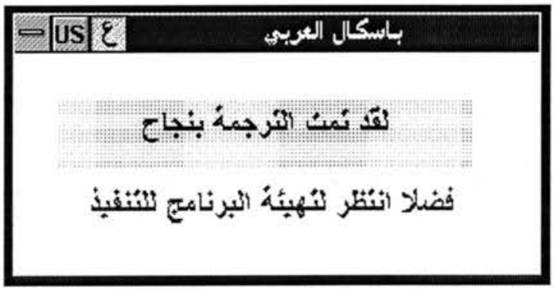
APPENDIX A

APE MESSAGES

This Appendix consists of the messages that are generated by the APE. There are three types of messages: informative messages, error messages, and dialog messages (require a response from the user). Each message will be shown here exactly as the APE displays it, along with the English translation and the main cause for generating this message.

A.1 INFORMATIVE MESSAGES

1.

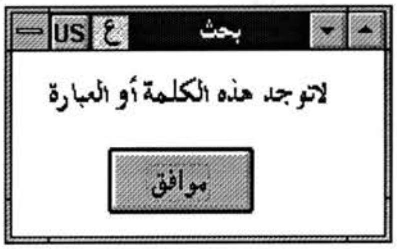


لقد تمت الترجمة بنجاح

فضلا انتظر لتهيئة البرنامج للتنفيذ

"Successful Compilation. Please wait for preparing the program for execution"
This message is displayed when the user compiles a program that is free of errors and the assembling and linking is in progress. This message will disappear automatically when the program is ready for execution.

2.

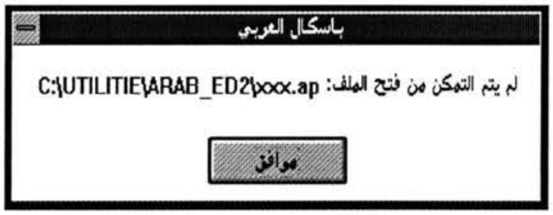


لاتوجد هذه الكلمة أو العبارة

"This word does not exist" (OK)
This message is displayed when the search for a word reaches the end or the beginning of the file without finding that word.

A.2 ERROR MESSAGES

1.

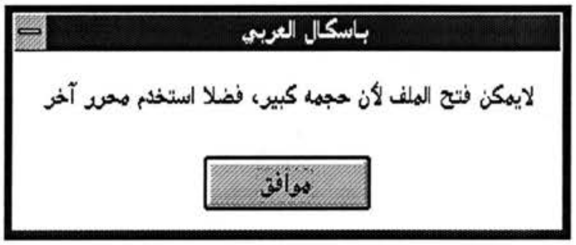


باسكال العربي

لم يتم التمكن من فتح الملف: C:\UTILITIE\ARAB_ED2\xxx.ap

"Unable to open the file <file-name>" (OK)
This message is displayed when the user tries to open a file that is not found in specified path.

2.

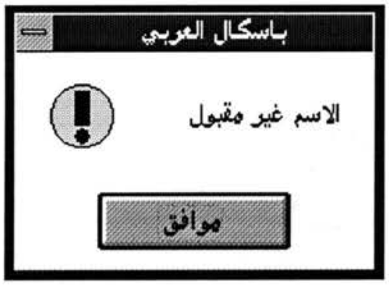


باسكال العربي

لايمكن فتح الملف لأن حجمه كبير، فضلا استخدم محرر آخر

"Unable to open the file because it is a big file. Please try another editor" (OK)
This message is displayed when the user tries to open a file that is more than 32K. The text window in Visual BASIC can hold only up to 32K.

3.

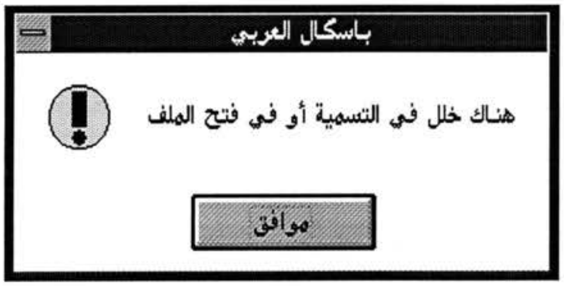


باسكال العربي

الاسم غير مقبول

"The name is not acceptable" (OK)
This message is displayed when the user enters some special characters (such as +, |, or ;) as part of the file name.

4.

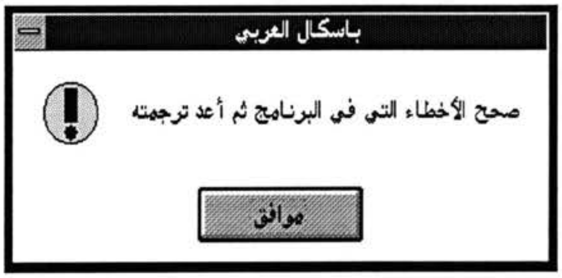


باسكال العربي

هناك خلل في التسمية أو في فتح الملف

"There is an error in the file name or in opening the file" (OK)
This message is displayed when the system cannot open the file due to corruption in the file name or overflow in the memory.

5.

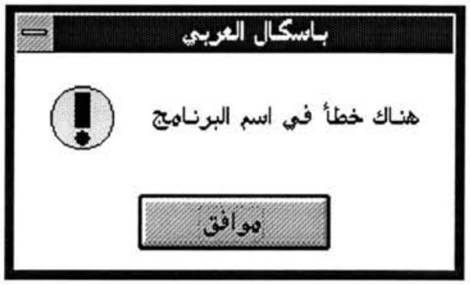


باسكال العربي

صحح الأخطاء التي في البرنامج ثم أعد ترجمته

"Correct the errors in the program then recompile it" (OK)
This message is displayed when the user tries to execute a program (by clicking the red traffic light) that has some compilation errors.

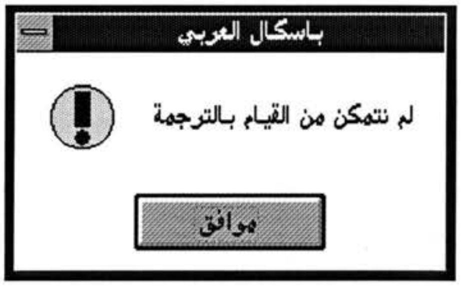
6.



هناك خطأ في اسم البرنامج

"There is an error in the file name" (OK)
This message is displayed when the user tries to compile a program that has not been saved (its name is still 'Untitled' or the compiler could not find the file name.)

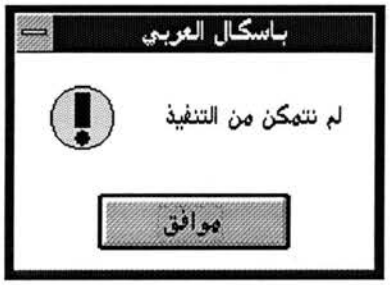
7.



لم نتمكن من القيام بالترجمة

"Unable to do the compilation" (OK)
This message is displayed when the compiler (APC) was not found in the expected path. (for example, if the user somehow deleted the compiler or altered its place in the directory).

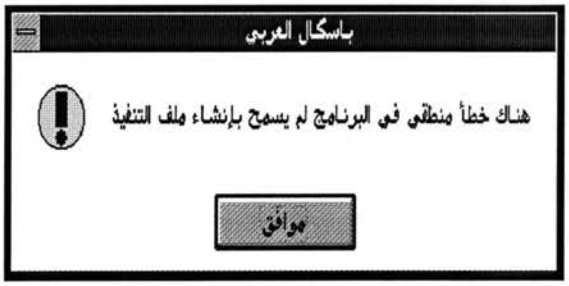
8.



لم نتمكن من التنفيذ

"Unable to do the execution" (OK)
This message is displayed when the *tasm* or *tlink* is not found in the expected place or the executable file was not found in the current directory.

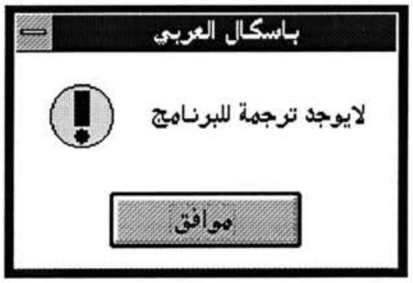
9.



هناك خطأ منطقي في البرنامج لم يسمح بإنشاء ملف التنفيذ

"There is a logical error that prevents the creation of the executable file" (OK)
This message is displayed when the compiler (APC) produces an assembly code that is not recognized by the Assembler (tasm) or the Linker (tlink). In this case, please notify the author.

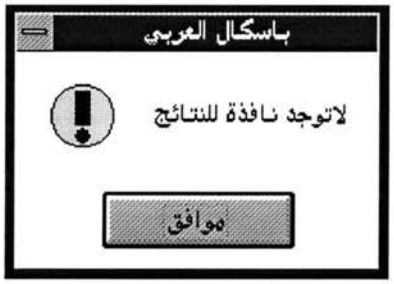
10.



لايوجد ترجمة للبرنامج

"There is no compilation for this program" (OK)
This message is displayed when the user tries to print the compilation window while there is no compilation window.

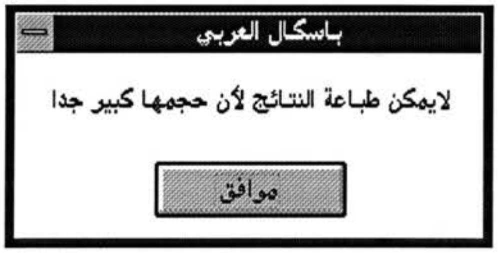
11.



لاتوجد نافذة للنتائج

"There is no execution window" (OK)
This message is displayed when the user tries to print the execution results while there is no execution window.

12.

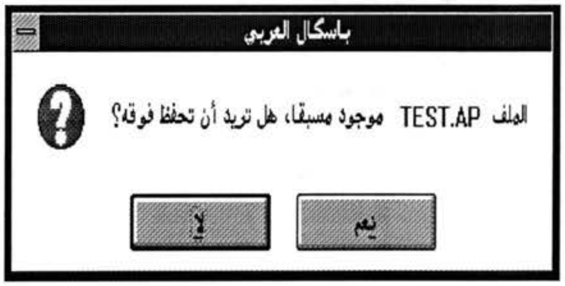


لايمكن طباعة النتائج لأن حجمها كبير جدا

"Unable to print the results because the output size is very large" (OK)
This message is displayed when the user tries to print a larger than 32KB file.

A.3 DIALOG MESSAGES

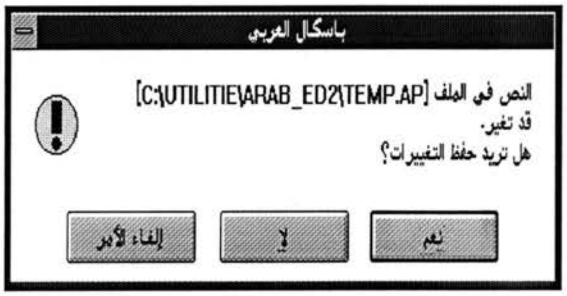
1.



الملف TEST.AP موجود مسبقا، هل تريد أن تحفظ فوقه؟

"The file <file-name> already exists. Do you want to overwrite it?" (YES, NO)
This message is displayed when the user tries to save a file by naming it with a name that already exists.

2.



النص في الملف [C:\UTILITIE\ARAB_ED2\TEMP.AP]
قد تغير.
هل تريد حفظ التغييرات؟

"The file <file-name> has been changed. Do you want to save it?" (YES, NO, CANCEL)
This message is displayed when the user tries to leave the APE or close a file without saving it.

APPENDIX B

LIST OF WORDS AND SYMBOLS IN THE ARABIC PASCAL

The following are the lists of reserved words, predefined types and identifiers, built-in functions and procedures, and symbols in English Pascal and their correspondences in Arabic Pascal.

B.1 RESERVED WORDS

AND	و		NIL	فارغ
ARRAY	مصفوفة، مصفوفه		NOT	ليس
BEGIN	ابدأ، ابدا		OF	من
CASE	حالة، حاله		OR	أو، او
CONST	ثابت		PACKED	مربوطة، مربوطه
DIV	قاسم		PROCEDURE	عملية، عمليه
DO	اعمل، إعمل		PROGRAM	برنامج
DOWNTO	نزول_إلى، نزول_الى		RECORD	سجل
ELSE	وإلا، والا		REPEAT	كرر
END	نهاية، نهايه		SET	مجموعة، مجموعه
FILE	ملف		THEN	فإن، فان
FOR	لقيم		TO	إلى، الى
FUNCTION	دالة، داله		TYPE	نوع
GOTO	اذهب_إلى، اذهب_الى		UNTIL	حتى
IF	إذا، اذا		VAR	متغير
IN	في، فى		WHILE	طالما

LABEL	عنوان		WITH	مع
MOD	باقي، باقى		FORWARD*	لاحق

* “FORWARD” is not a reserved word; it is the only directive in Pascal Language.

B.2 PREDEFINED IDENTIFIERS AND TYPES

INPUT	مدخل		INTEGER	صحيح
OUTPUT	مخرج		BOOLEAN	منطقي، منطقى
CHAR	حرف		REAL	حقيقي، حقيقى
FALSE	خطأ، خطا		TEXT	نص
TRUE	صواب			

B.3 BUILT-IN FUNCTIONS AND PROCEDURES

(1) Functions

ABS	مطلق		ORD	ترتيب
ARCTAN	قوظا		PRED	سابق
CHR	حرفي، حرفى		ROUND	تقريب
COS	جتا		SIN	جا
EOF	نها_م		SQR	مربع
EOLN	نها_س		SQRT	جذر_٢
EXP	أس، اس		SUCC	تابع
LN	لو		TRUNC	بتر
ODD	فردي، فردى			

(2) Procedures

DISPOSE	تخلص		RESET	قراءة، قراءه
NEW	جديد		REWRITE	كتابة، كتابة
READ	اقرأ، اقر ا		WRITE	اكتب
READLN	اقرأ_س، اقرا_س		WRITELN	اكتب_س

Note: Procedures GET, PUT, and PAGE are not implemented in the Arabic Pascal.

B.4 SYMBOLS

The following are the symbols that have been changed in the Arabic Pascal. The symbols that are not mentioned here are the same in both English Pascal and Arabic Pascal. The character “E” is the E-notation in real numbers.

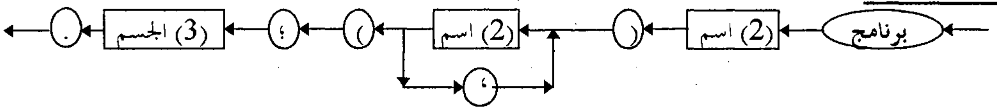
English	Arabic		English	Arabic		English	Arabic
‘	“		,	،		>	<
(	)		)	(		<=	=>
;	؛		E	س		>=	=<
{	}		}	{		]	[
:=	=:		<	>		[	]

APPENDIX C

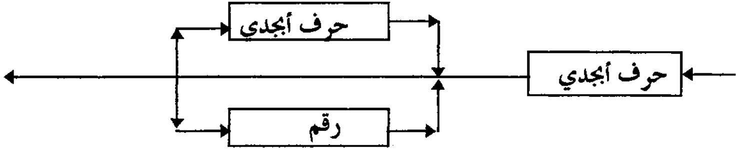
SYNTAX DIAGRAMS FOR THE ARABIC PASCAL

The following are the syntax diagrams which precisely define the syntax of the Arabic Pascal. The usual conventions of syntax diagrams are assumed. The circles stand for terminal symbols while the rectangles stand for non-terminal symbols. The last two pages of this Appendix give the syntax diagrams of the built-in functions and procedures.

(1) الهيكل العام للبرنامج



(2) اسم



ملحوظات:

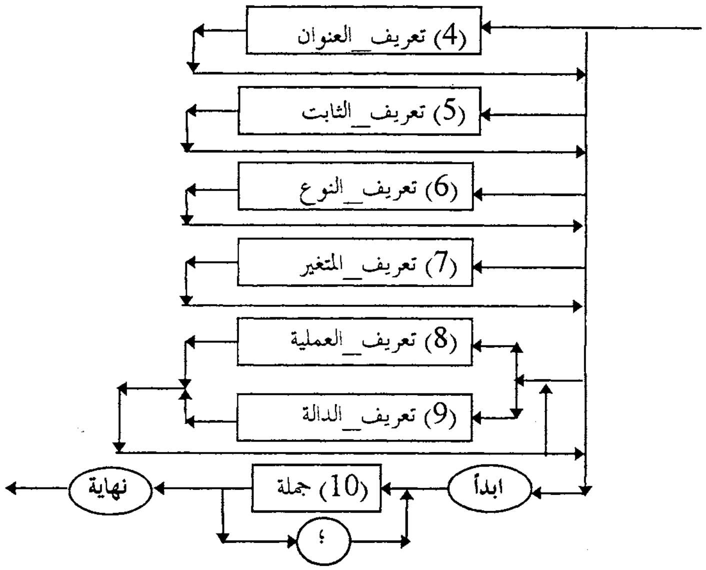
(1) حرف أبجدي هو ما بين حرف الهمزة وحرف الياء بالإضافة للشرطة التحتية "_".

(2) يجب ألا يتخلل الاسم أي فراغ.

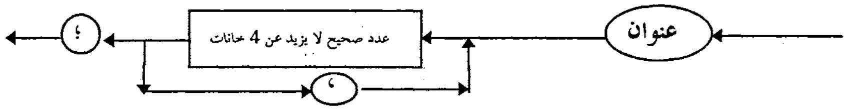
(3) الاسم محدود بطول 80 حرف.

(4) رقم هو ما بين صفر و 9.

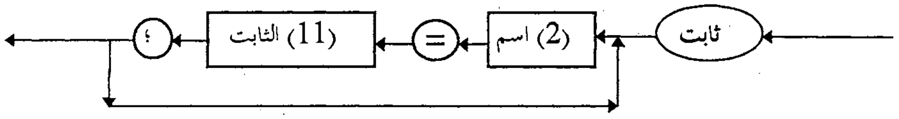
(3) الجسم



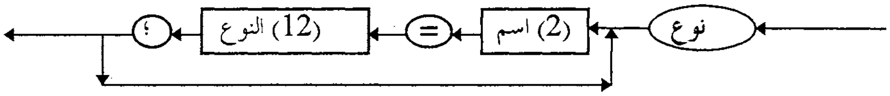
(4) تعريف_العنوان



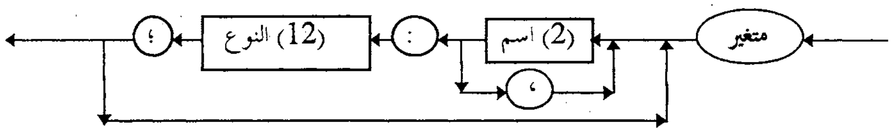
(5) تعريف_الثابت



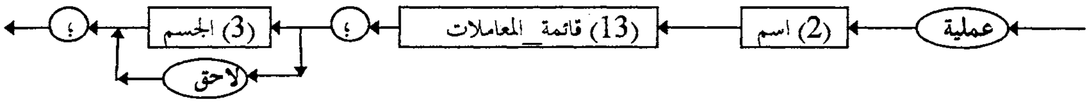
(6) تعريف_النوع



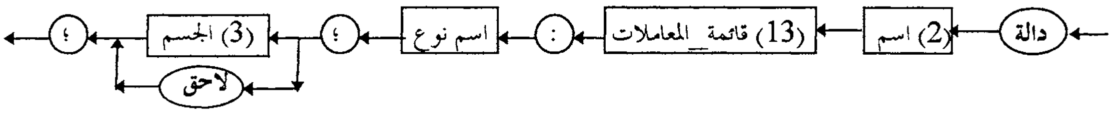
(7) تعريف_المتغير



(8) تعريف_العملية

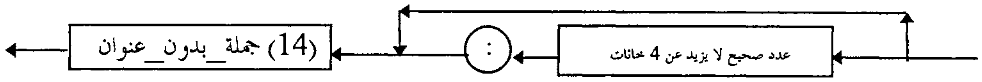


(9) تعريف_الدالة

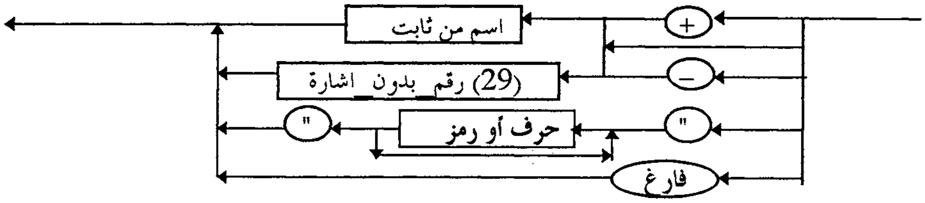


ملحوظة: "اسم نوع" تعني أي اسم معرّف في "نوع" أو أي نوع من الأنواع المعروفة وهي "صحيح"، "حقيقي"، "منطقي"، "حرف".

(10) جملة



(11) الثابت

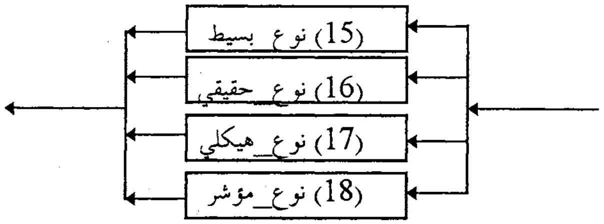


ملحوظات:

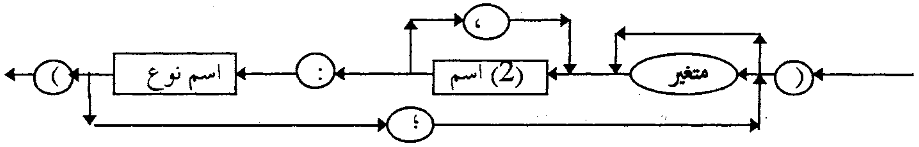
(1) "اسم من ثابت" تعني أي اسم تم تعريفه بأنه ثابت.

(2) "حرف أو رمز" يجب ألاّ يحتوي على الرمز "، إلا أن يكون "" ويجب أن يكون النص في سطر واحد فقط.

(12) النوع

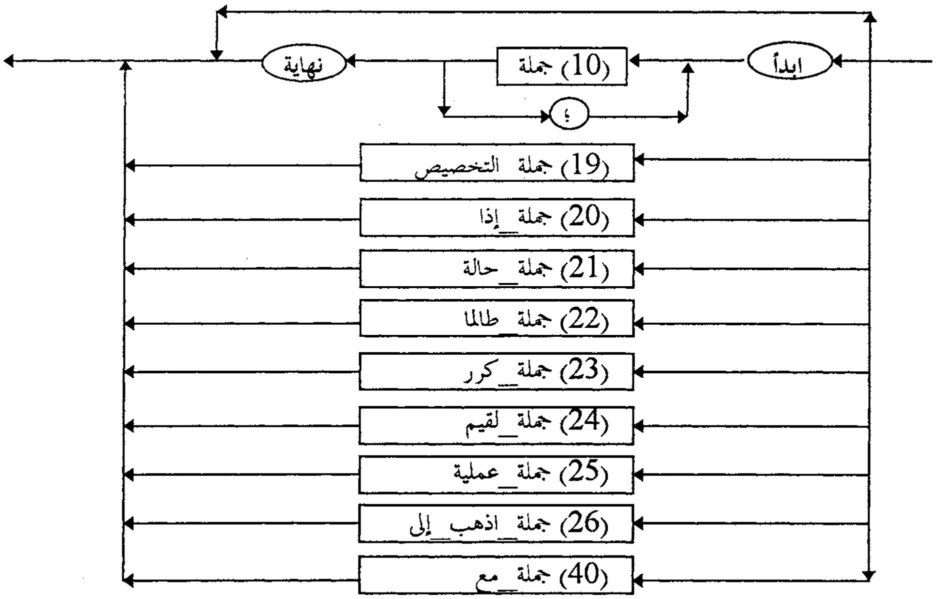


(13) قائمة_المعاملات

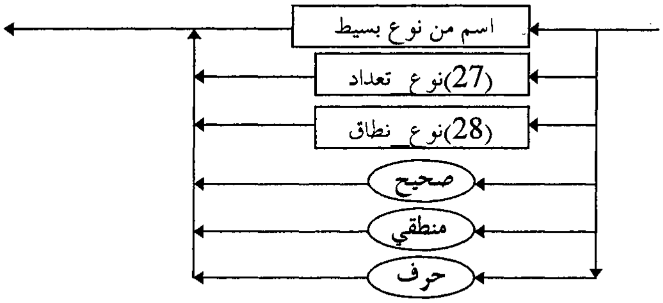


ملحوظة: "اسم نوع" تعني أي اسم معرّف في "نوع" أو أي نوع من الأنواع المعروفة وهي "صحيح"، "حقيقي"، "منطقي"، "حرف".

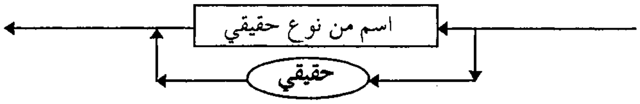
(14) جملة_بدون_عنوان



(15) نوع_بسيط



(16) نوع_حقيقي

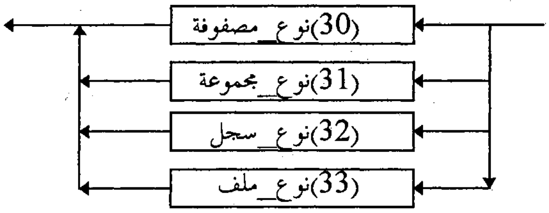


ملحوظة: "اسم من نوع حقيقي" تعني أي اسم معرّف في قسم "نوع" على أنه "حقيقي".

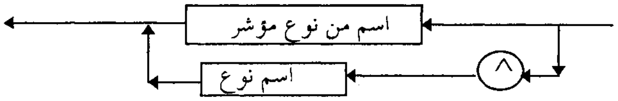
مثال: نوع عدد = حقيقي؛

متغير س : عدد؛

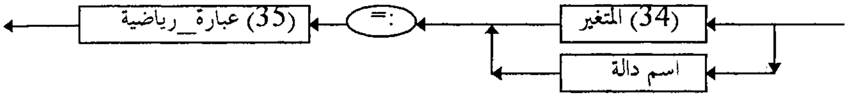
(17) نوع_هيكلي



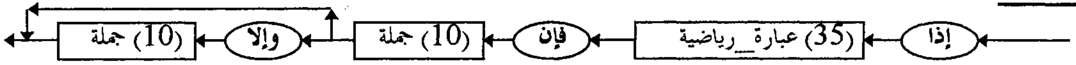
(18) نوع_مؤشر



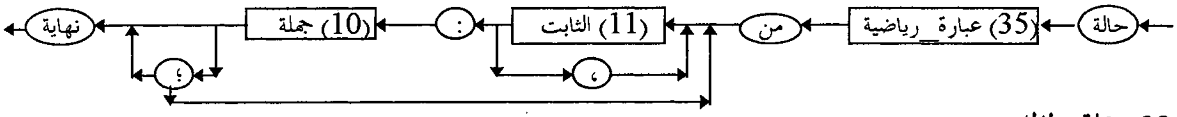
(19) جملة_التخصيص



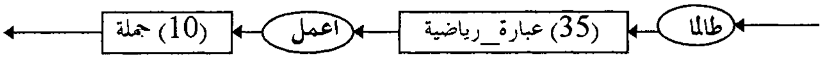
(20) جلة_إذا



(21) جملة_حالة

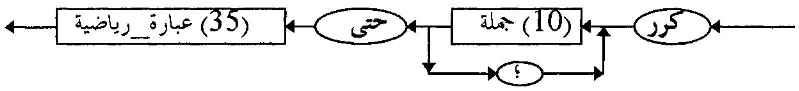


(22) جملة_طالما



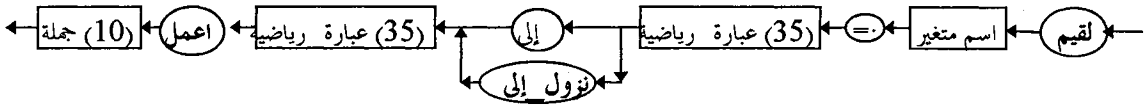
ملحوظة: العبارة الرياضية هنا لابد أن تكون نتيجتها من نوع "منطقي".

(23) جملة_كرر

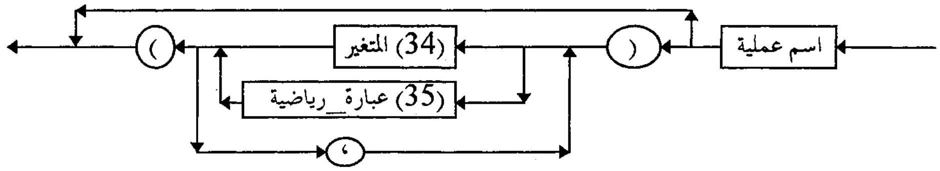


ملحوظة: العبارة الرياضية هنا لابد أن تكون نتيجتها من نوع "منطقي".

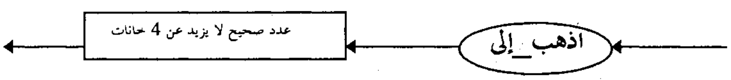
(24) جملة_لقيم



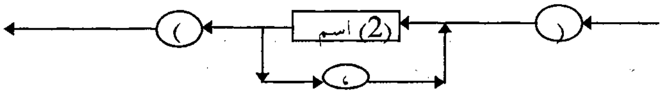
(25) جملة_عملية



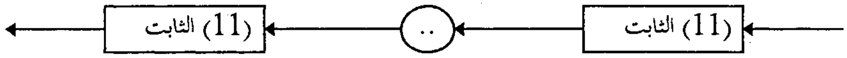
(26) جملة_اذهب_إلى



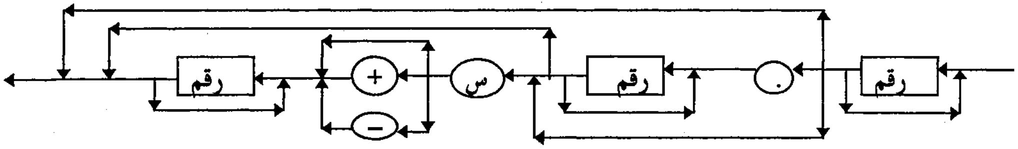
(27) نوع_تعداد



(28) نوع_نطاق

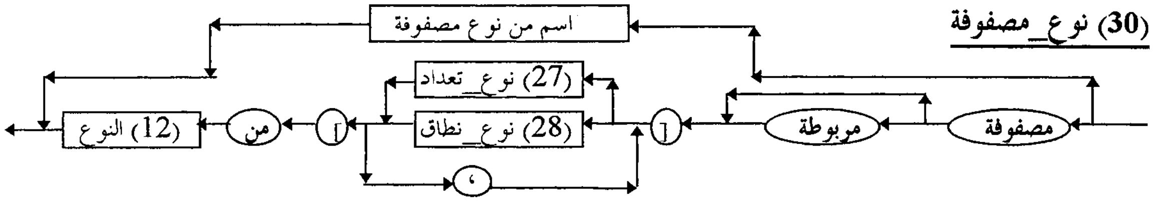


(29) رقم_بدون_اشارة

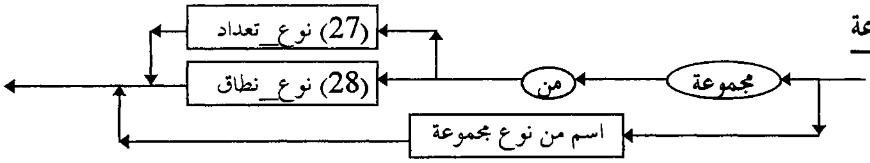


ملحوظة: النقطة في العدد الكسري تتحول آليا إلى فاصلة عشرية "," عند استخدام الأعداد الهندية (١،٢،٣، ... ٩).

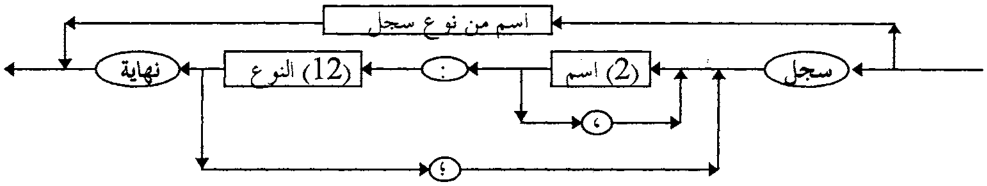
(30) نوع_مصفوفة



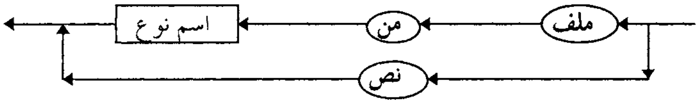
(31) نوع_مجموعة



(32) نوع_سجل



(33) نوع_ملف

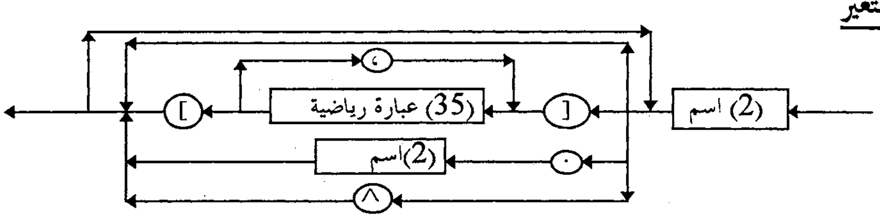


ملحوظات:

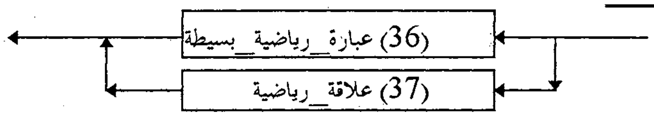
(1) "اسم نوع" يجب أن تكون من نوع صحيح، حقيقي، أو حرف.

(2) "نص" هو اسم معرّف مسبقا في لغة باسكال على النحو التالي: نص = ملف من حرف؛

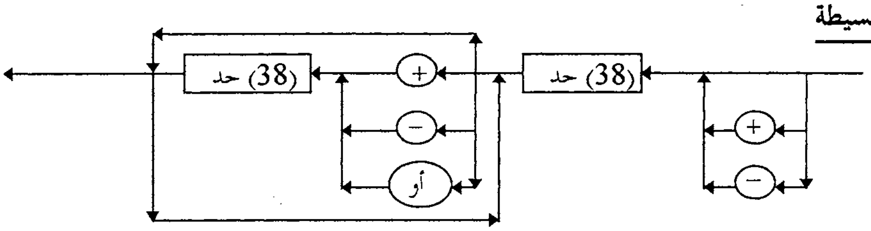
(34) المتغير



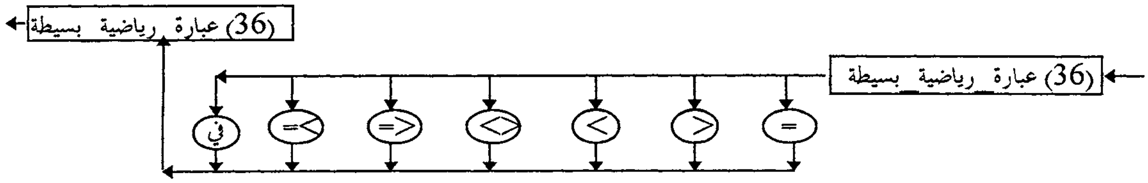
(35) عبارة_رياضية



(36) عبارة_رياضية_بسيطة

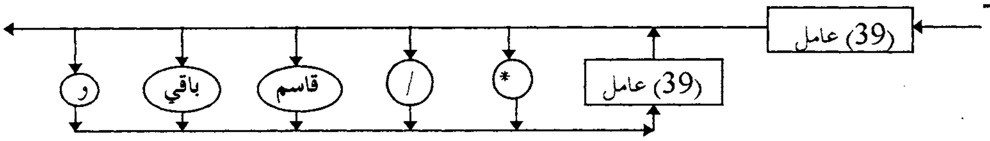


(37) علاقة_رياضية

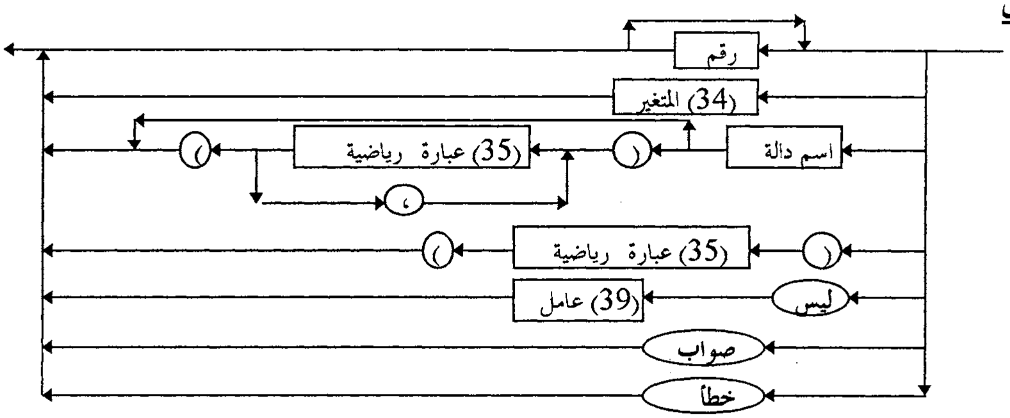


ملحوظة: المجموعة لها خصائص خاصة. انظر رقم (42) و (43) و (44).

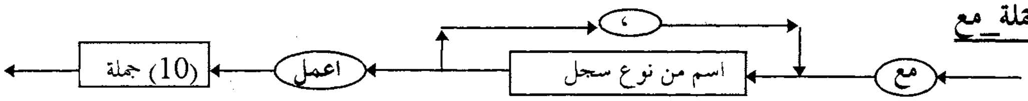
(38) حد



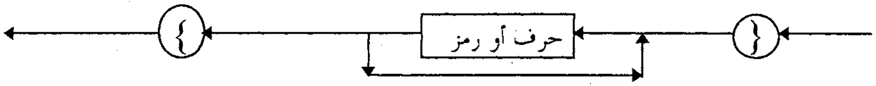
(39) عامل



(40) جملة_مع

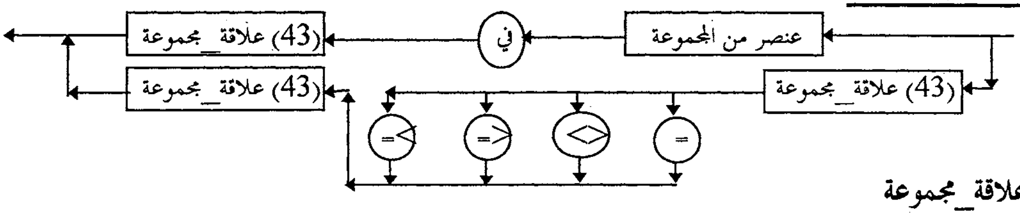


(41) التعليق

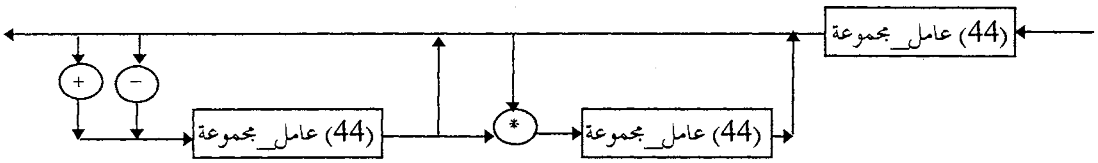


ملحوظة: "حرف أو رمز" يجب ألاّ يحتوي على الرمز }.

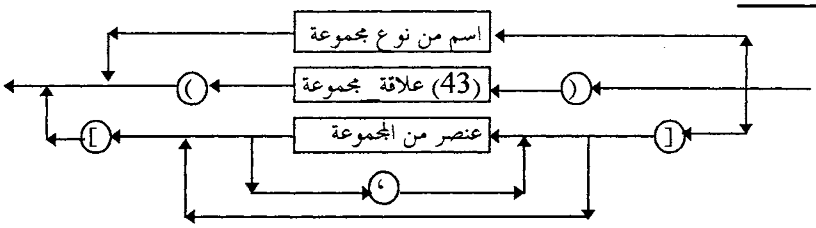
(42) العلاقة في المجموعة



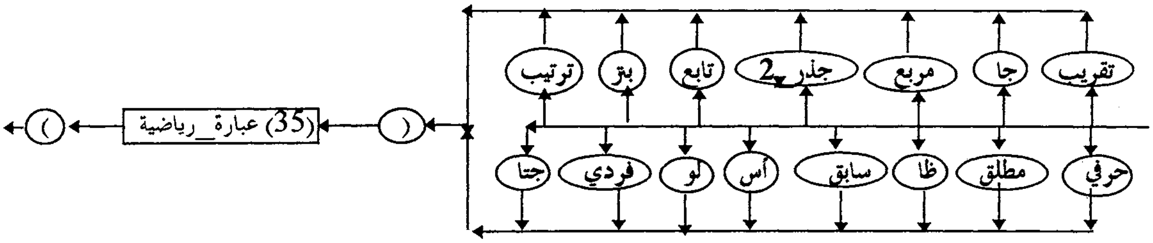
(43) علاقة_مجموعة



(44) عامل_مجموعة



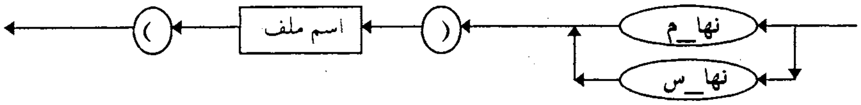
(45) الدوال الرياضية



ملحوظة:

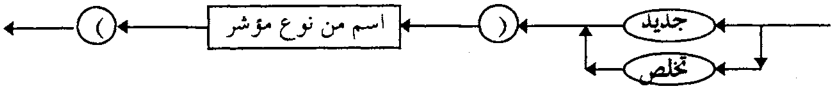
يجب أن تتوافق العبارة الرياضية مع التعريف المسبق للدالة. انظر التفصيل في القسم الخاص بالدوال المعرفة في لغة باسكال العربية (Section 5.2.4.6).

(46) الدوال الأخرى:

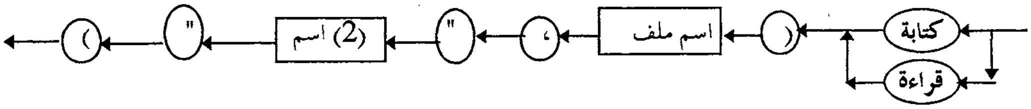


ملحوظة: اسم الملف لابد أن يكون من نوع "نص" في حالة استخدام "نها_سطر".

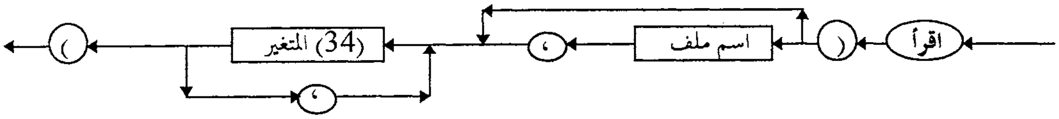
(47) العملية: جديد و تخلص



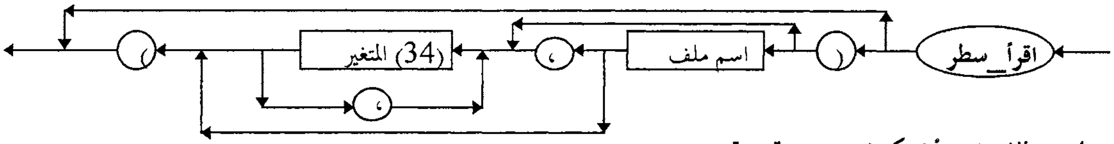
(48) العملية: قراءة وكتابة



(49) العملية: اقرأ

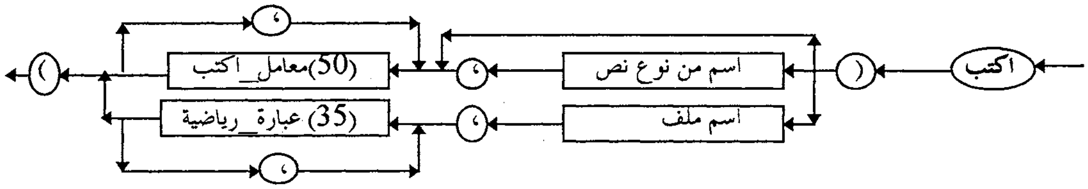


(50) العملية: اقرأ_سطر

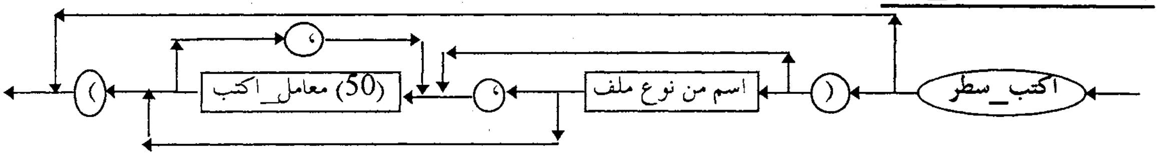


ملحوظة: الملف يجب أن يكون من نوع "نص".

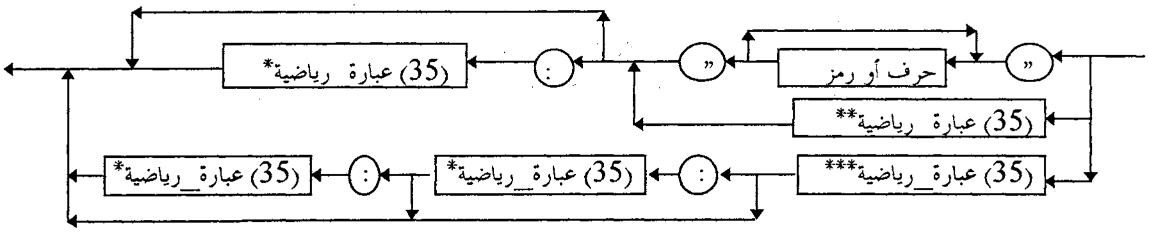
(51) العملية: اكتب



(52) العملية: اكتب_سطر



(53) معامل_اكتب_سطر



ملحوظة:

* نتيجة العبارة الرياضية لابد أن تكون من نوع "صحيح".

** نتيجة العبارة الرياضية لابد أن تكون من نوع "صحيح" أو "منطقي".

*** نتيجة العبارة الرياضية لابد أن تكون من نوع "حقيقي".

APPENDIX D

ERROR MESSAGES

The following is a list of the error messages along with their English translation. There are two types of errors related to the Arabic Pascal language. The first type of errors is the compiler generated error messages which are caused by syntax errors in the user program. The second type of errors is the run-time error messages which occur during the execution.

D.1 COMPILER ERROR MESSAGES

Cannot open the source file	لم يتم التمكن من فتح ملف البرنامج
Unexpected end of file	نهاية غير متوقعة في ملف البرنامج
Invalid number	الرقم غير مقبول
Invalid fraction	الجزء الكسري غير مقبول
Invalid exponent	الجزء الأسي غير مقبول
Too many digits in the number	عدد الخانات في الرقم كثيرة جدا
Outside the real range	الرقم خارج النطاق المسموح به للرقم الحقيقي
Outside the integer range	الرقم خارج النطاق المسموح به للرقم الصحيح
Missing left parenthesis	القوس الأيسر '(' مفقود
Missing right parenthesis	القوس الأيمن ’)’ مفقود
Invalid expression	الصيغة الرياضية غير مقبولة
Invalid assignment statement	جملة التخصيص ':=' غير مقبولة
Missing identifier	المتغير مفقود
Missing assignment symbol :=	علامة التخصيص ':=' مفقودة

Undefined identifier	متغير غير معرّف
Invalid statement	الجملة غير مقبولة
Unexpected symbol	حرف أو متغير غير متوقع في هذا الموقع
Missing PROGRAM word	كلمة 'برنامج' مفقودة
Missing BEGIN word	كلمة 'ابدأ' مفقودة
Missing semicolon ;	الفاصلة المنقوطة '؛' مفقودة
Missing DO word	كلمة 'اعمل' مفقودة
Missing UNTIL word	كلمة 'حتى' مفقودة
Missing THEN word	كلمة 'فإن' مفقودة
Invalid FOR control variable	المتغير الذي يتحكم في عملية التكرار غير مقبول
Missing OF word	كلمة 'من' مفقودة
Invalid constant	الثابت غير مقبول
Missing constant	الثابت مفقود
Missing colon :	الحرف ':' مفقود
Missing END word	كلمة 'نهاية' مفقودة
Missing TO or DOWNTO word	كلمة 'إلى' أو 'نزول_إلى' مفقودة
The identifier has been defined before	لقد تم تعريف هذا المتغير سابقا
Missing equal sign =	الحرف '=' مفقود
Invalid type	النوع غير مقبول
Not a type identifier	ليس المتغير من الأنواع المعرفة سابقا
Invalid subrange type	النطاق المعرف غير مقبول
Not a constant identifier	يجب أن يكون الرمز ثابتا وليس متغيرا
Missing range symbol ..	النقطتان '..' مفقودتان
Mismatch in types	الأنواع غير متوافقة
Invalid identifier usage	استخدام خاطئ للمتغير
Incompatible assignment	الطرفان ليسا متوافقين في النوع
Min. value greater than max. value	الحد الأدنى أكبر من الحد الأكبر
Missing]	الحرف ']' مفقود
Missing [	الحرف '[' مفقود
Invalid index type	نوع دليل المصفوفة غير مقبول

Missing period	الحرف '.' مفقود
Too many subscripts	عدد الأدلة للمصفوفة كثيرة
Invalid field in the record	الحقل في السجل غير مقبول
Nesting too deep	عدد الجمل المتداخلة كثيرة
Already specified in FORWARD	لقد تم تحيدها سابقا في الجملة التي تحتوي كلمة 'لاحق'
Wrong number of actual parameters	عدد المعاملات الحقيقية في الدالة أو العملية غير صحيح
Invalid VAR parameter	المعامل لابد أن يكون متغير وليس ثابت
Not a record variable	المتغير ليس من نوع سجل
Missing variable	المتغير مفقود
Code segment overflow	لقد امتلأ القسم الخاص بالبرنامج في الذاكرة
Stack overflow	لقد امتلأ الستاك
The label is invalid or missing: expecting an integer between 0 and 9999	العنوان خاطئ أو مفقود: متوقع عدد صحيح بين ٠ و ٩٩٩٩
The label is not defined	العنوان غير معرَف داخليا
No label corresponding to GOTO	لا يوجد عنوان مطابق لجملة ‘اذهب_إلى‘
The label has been used before	لقد تم استخدام هذا العنوان سابقا
The set has more than 16 elements	المجموعة تحتوي على أكثر من ١٦ عنصر
File must be of integer, real, or char type	الملف لابد أن يكون من نوع صحيح أو حقيقي أو حرف
The left double-quote is missing	علامة التنصيص تم فتحها و لم يتم غلقها
The pointer is not defined	المؤشر غير معرف
The variable is not of file type	المتغير ليس من نوع ملف
The file name is missing	اسم الملف غير موجود
The file has not been opened for reading	لم يتم فتح الملف للقراءة
The file has not been opened for writing	لم يتم فتح الملف للكتابة
The variable is not of pointer type	المتغير ليس من نوع مؤشر
The pointer symbol ‘^” is missing	علامة المؤشر ‘^‘ مفقودة
Relations ‘>‘ and ‘<‘ are not accepted in sets	العلاقة ‘>‘ و ‘<‘ غير مقبولة في المجموعات
The expression is not valid in sets	العملية الحسابية ليست مقبولة بين المجموعات
One of the variables is not of set type	أحد المتغيرات ليس من نوع مجموعة
Too many syntax errors	الأخطاء كثيرة جدا

D.2 RUN-TIME ERROR MESSAGES

Run-time error messages cause the user program to display an error message and terminate. Run-time errors do not include all errors that may occur during execution. There are two types of errors that are not reported by the Arabic Pascal. The first is the errors that need a considerable overhead in the compiler generated code to report them , while they are not fatal errors (from the system point of view). Examples of this type are an integer operation that may cause an out of range result and out of range index values. Both Turbo Pascal and Turbo C do not perform such checking in their default mode [BTP89, MSF88]. In this case, it is the responsibility of the programmer to ensure the correctness of usage. The second type is the fatal errors that are related to the MS-Window functions. When such an error happens, the MS-Windows takes over any of its application to report a system error. Examples of this type are out of memory to load the execution code and accessing a protected area in the MS-Windows. The MS-Windows (including the Arabic versions) display these errors in English. The following is a list of possible run-time error messages that may occur during the execution.

The entered number is not an integer	الرقم الذي تمت قراءته ليس عددا صحيحا
The entered number is out of the integer range	الرقم الذي تمت قراءته خارج نطاق العدد الصحيح
The entered number is not a real	الرقم الذي تمت قراءته ليس عددا حقيقيا
The entered number is out of the real range	الرقم الذي تمت قراءته خارج نطاق العدد الحقيقي
Error in real number operation (out of range)	خطأ في الأعداد الحقيقية (خارج النطاق)
Cannot open the file for reading	لم يتم التمكن من فتح الملف للقراءة
Cannot open the file for writing	لم يتم التمكن من فتح الملف للكتابة
Out of memory	لم يتبق شئ من الذاكرة
Divide by zero	قسمة على صفر
Negative number in the square root	الرقم تحت الجذر التربيعي سالب
Out of the set limits	أحد العناصر خارج نطاق المجموعة
Abnormal Program Termination	البرنامج انتهى بشكل غير طبيعي
Stack Overflow	الستاك قد تجاوز الحد الأعلى

APPENDIX E

TEST PROGRAMS

In Chapter 5, Section 5.5, I gave two examples of Arabic Pascal programs along with their corresponding English Pascal programs. In this Appendix, I will give some test programs translated directly from a well-known book "Pascal User Manual and Report" [JW74]. These programs are used to test the APC. At the beginning of each example, I will give the example number and page number in that reference. At the end of each program, I will also give some sample input and the output. The sample input is usually same as the one provided in that book. In some programs, the code for reading the input is added if the original program assumes the input has been already read. Note that some real numbers are printed by the APC as fixed point real numbers while they are printed in the reference using the E-notation. The reason is that the APC selects the format that produces the shortest output (similar to the %g specifier in C language).

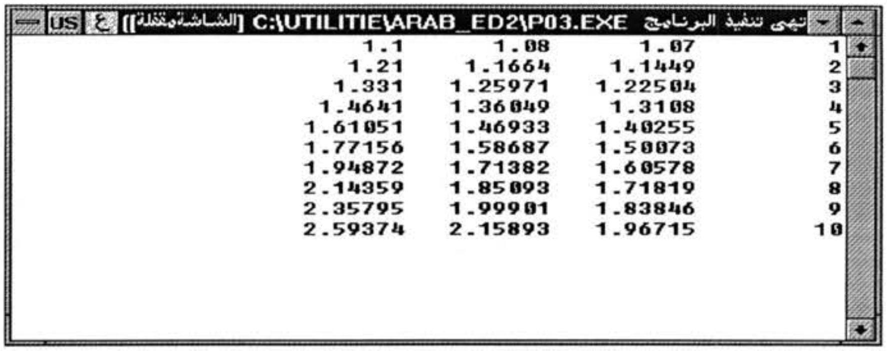
The last example, E.23, complements the other programs in utilizing some concepts that are not examined in that book. For example, the book does not have a complete program that explains the *case* statement.

1. Program 0.1 (Page 3)

It Computes the annual inflation rates of 7, 8, 10 per cent for 10 years.

This program mainly tests constant definition, integer and real variables, simple statements, and *repeat* statement.

```
{برنامج ٠ . ١ صفحة ٣}
{حساب التضخم }

برنامج التضخم (مخرج)؛
ثابت ن = ١٠؛
متغير ع : صحيح؛
      ت١، ت٢، ت٣ : حقيقي؛
ابدأ  ع := ٠؛  ت١ := ١,٠؛  ت٢ := ١,٠؛ ت٣ := ١,٠؛
كرر   ع := ع + ١؛
      ت١ := ت١ * ١,٠٧؛
      ت٢ := ت٢ * ١,٠٨؛
      ت٣ := ت٣ * ١,١٠؛
      اكتب_س(ع، ت١، ت٢، ت٣)؛
حتى   ع = ن
نهاية.
```


تهى تنفيذ البرنامج C:\UTILITIE\ARAB_ED2\P03.EXE [الشاشةمقفلة]
ع
US

```
      1.1       1.08       1.07        1
     1.21     1.1664     1.1449        2
    1.331    1.25971    1.22504        3
   1.4641    1.36049     1.3108        4
  1.61051    1.46933    1.40255        5
  1.77156    1.58687    1.50073        6
  1.94872    1.71382    1.60578        7
  2.14359    1.85093    1.71819        8
  2.35795    1.99901    1.83846        9
  2.59374    2.15893    1.96715       10
```

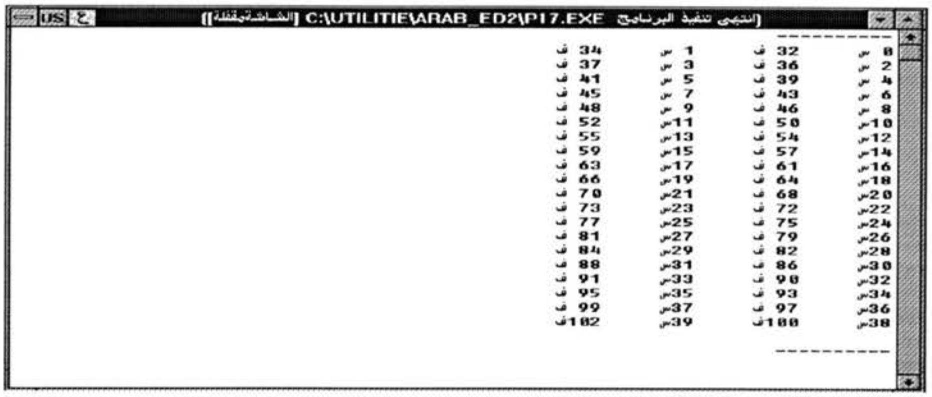
2. Program 3.1 (Page 17)

Example of constant definition. It converts the Celsius degree to Fahrenheit.

This program mainly tests constant definitions: integer, real, and string, subrange variables, *for* statement, *if* statement, and *round* and *odd* functions.

```
{برنامج ٣ . ١ صفحة ١٧}
{مثال على تعريف الثوابت}

برنامج تحويل (مخرج)؛
ثابت  اجمع = ٣٢؛
      اضرب = ١,٨؛
      أدنى = ٠؛
      أعلى = ٣٩؛
      فاصل = "----------"؛
متغير
      درجة : أدنى .. أعلى؛
ابدا
      اكتب_س(فاصل)؛
      لقيم  درجة := أدنى  إلى  أعلى  اعمل
        ابدا
          اكتب(درجة :٢، "س        "، تقريب(درجة*اضرب + اجمع):٣، "ف        ")؛
          اذا  فردي(درجة)    فان  اكتب_س؛
```

```
نهاية ؛
  اكتب_س؛
  اكتب_س(فاصل) ؛
نهاية .
```


[الشاشة مقفلة] C:\UTILITIE\ARAB_ED2\P17.EXE انتهى تنفيذ البرنامج

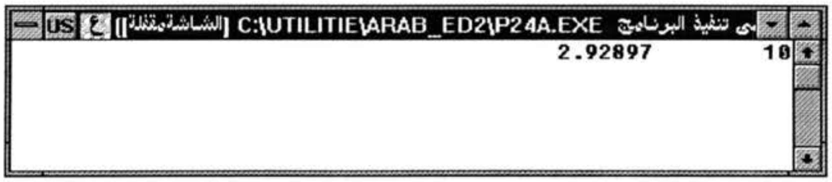
```
                                           ----------
ف 34      س 1       ف 32       س 0
ف 37      س 3       ف 36       س 2
ف 41      س 5       ف 39       س 4
ف 45      س 7       ف 43       س 6
ف 48      س 9       ف 46       س 8
ف 52      س11       ف 50       س10
ف 55      س13       ف 54       س12
ف 59      س15       ف 57       س14
ف 63      س17       ف 61       س16
ف 66      س19       ف 64       س18
ف 70      س21       ف 68       س20
ف 73      س23       ف 72       س22
ف 77      س25       ف 75       س24
ف 81      س27       ف 79       س26
ف 84      س29       ف 82       س28
ف 88      س31       ف 86       س30
ف 91      س33       ف 90       س32
ف 95      س35       ف 93       س34
ف 99      س37       ف 97       س36
ف102      س39       ف100       س38

                                           ----------
```

3. Program 4.4 (Page 24)

Compute *h(n)* = *1* + *1/2* + *1/3* + . . . + *1/n*. Examples 4.1 to 4.3 are trivial programs that test compound, *while*, and *repeat* statements, respectively. These statements are tested in other examples also.

This program mainly tests *for* statement with *downto*.

```
{برنامج ٤ . ٤ صفحة ٢٤}
{حساب د (ن) = ١ + ١/٢ + ١/٣ + .... + ١/ن }

برنامج حساب(مدخل، مخرج) ؛
متغير ع ، ن : صحيح؛
      ح : حقيقي؛
ابدأ ن := ١٠؛ اكتب(ن:١٠) ؛
      ح := ٠؛
      لقيم ع := ن نزول_إلى ١ اعمل
            ح := ح + ١/ع؛
      اكتب_س(ح) ؛
نهاية .
```


[الشاشة مقفلة] C:\UTILITIE\ARAB_ED2\P24A.EXE ى تنفيذ البرنامج

```
2.92897        10
```

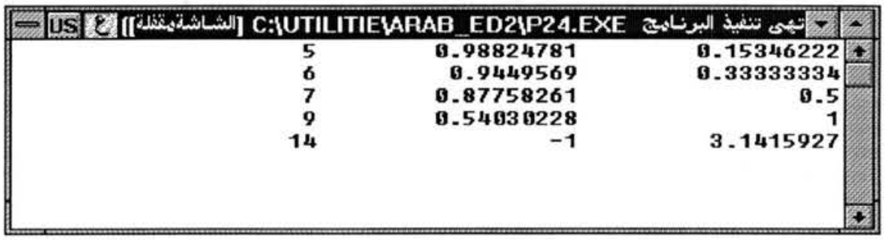
4. Program 4.5 (Page 24)

Computing cosine using the expansion:

cos(x) = *1* - $x^2/(2*1)$ + $x^4/(4*3*2*1)$ - .....

This program mainly tests *for* statement, *while* statement, *abs* and *sqr* functions, and reading from a file of real numbers. The original program reads from the standard input.

```
{برنامج ٤ . ٥، صفحة ٢٤}
{حساب قيمة "جتا" باستخدام
جتا (س) = ١ - س**٢/(٢*١) + س**٤/(٤*٣*٢*١) - . . .}

برنامج جتا_س(مدخل، مخرج)؛
ثابت ابسلون = ١س-١٤؛
متغير
   س، مربع_س، مجموع، ت : حقيقي؛
   ل، ك، ن : صحيح؛
   م : ملف من حقيقي؛
ابدأ
   قراءة(م ، "cosine.inp") ؛
   اقرأ (م ،ن) ؛
   لقيم ل :=١ إلى ن اعمل
   ابدأ اقرأ (م ،س) ؛
       ت :=١؛
       ك := ٠؛
       مجموع := ١؛
       مربع_س := مربع (س) ؛
       طالما مطلق(ت) > ابسلون* مطلق(مجموع) اعمل
       ابدأ ك := ك + ٢؛
           ت := -ت * مربع_س / (ك*(ك-١)) ؛
           مجموع := مجموع + ت؛
       نهاية؛
       اكتب_س(س:١٨ : ٨، مجموع: ١٨ : ٨ ، ك قاسم ٢ : ٨)
   نهاية
نهاية.
```


US
ع
تهى تنفيذ البرنامج C:\UTILITIE\ARAB_ED2\P24.EXE [الشاشةمقفلة]

```
                     5      0.98824781       0.15346222
                     6       0.9449569       0.33333334
                     7      0.87758261              0.5
                     9      0.54030228                1
                    14              -1        3.1415927
```

5. Program 4.6 (Page 25)

Compute *1 - 1/2 + 1/3 - . . . + 1/9999 - 1/10000* in four different ways.

1) left to right, in succession

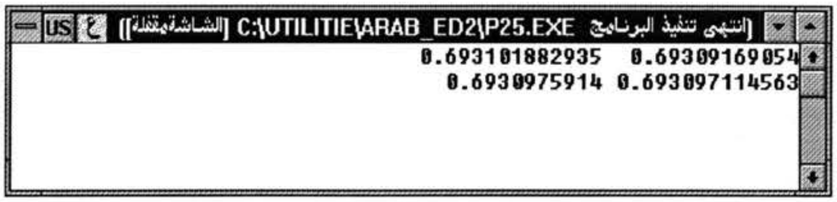
2) left to right, all positive and negative terms, then subtract

3) right to left in succession

4) right to left, all positive and negative terms, then subtract

This program mainly tests *for* statement.

```
{برنامج ٤ . ٦  صفحة  ٢٥}
{حساب  ١- ١/٢ + ١/٣ - ... + ١/٩٩٩٩ - ١/١٠٠٠٠. بأربعة طرق:
١ .من اليمين إلى اليسار بالتتابع
٢ .من اليمين إلى اليسار بجمع الموجبات على حدة وكذلك السالبات
   ثم الطرح.
٣ .مثل رقم ١ ولكن من اليسار إلى اليمين.
٤ .مثل رقم ٢ ولكن من اليسار إلى اليمين.}

برنامج  جمع (مخرج)؛
متغير
 م١، م٢م ، م٢س، م٣، م٤م ، م٤س، يمين_م ،
 يمين_س، يسار_م ، يسار_س : حقيقي؛
 ع : صحيح؛
ابدأ
 م١ := ٠,٠؛ م٢م := ٠,٠؛ م٢س := ٠,٠؛ م٣:= ٠,٠؛
 م٤م := ٠,٠؛ م٤س := ٠,٠؛
 لقيم  ع:= ١  إلى ٥٠٠٠  اعمل
 ابدأ
   يمين_م := ١ / (٢*ع -١)؛  { الحدود الموجبة. من اليمين}
   يمين_س := ١ / (٢*ع)؛  { الحدود السالبة. من اليمين }
   يسار_م := ١ / (١٠٠٠١ - ٢*ع)؛ { الحدود الموجبة. من اليسار}
   يسار_س := ١ / (١٠٠٠٢ - ٢*ع)؛ { الحدود السالبة. من اليسار}
   م١ := م١ + يمين_م - يمين_س؛
   م٢م := م٢م + يمين_م ؛   م٢س := م٢س + يمين_س؛
   م٣ := م٣ + يسار_م - يسار_س؛
   م٤م := م٤م + يسار_م ؛   م٤س := م٤س + يسار_س؛
 نهاية؛
 اكتب_س(م١ : ١٥ : ١٢، م٢م-م٢س : ١٥ : ١٢)؛
 اكتب_س(م٣ : ١٥ : ١٢، م٤م-م٤س : ١٥ : ١٢)؛
نهاية.
```


[انتهى تنفيذ البرنامج C:\UTILITIE\ARAB_ED2\P25.EXE [الشاشةمقفلة]]
US
ع

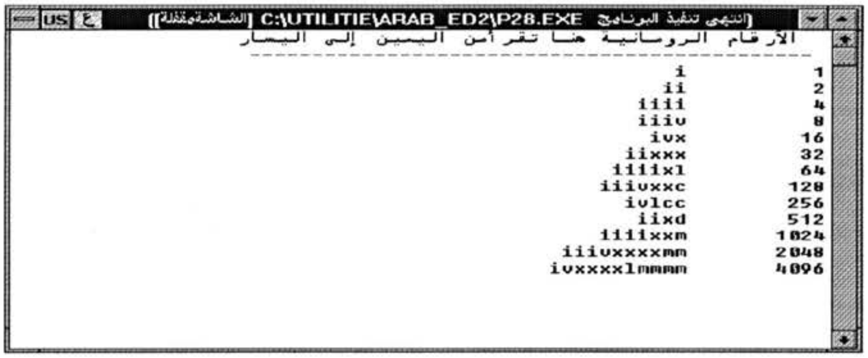
```
0.693101882935   0.69309169054
 0.6930975914 0.693097114563
```

6. Program 4.7 (Page 28)

Write Roman numerals. Note that the printed Roman numbers are read from right to left in the output. Examples 4.8 and 4.9 will be skipped because examples 11.8 and 6.2, respectively, are expansions for these two examples.

This program mainly tests *repeat, while, and if* statements.

```
{برنامج ٤ . ٧ صفحة ٢٨}
{كتابة الأرقام الرمانية }

برنامج روماني(مخرج) ؛
متغير       س،ص : صحيح؛
ابدا
  ص := ١ ؛
  اكتب_س(" الأرقام الرومانية هنا تقرأمن اليمين إلى اليسار ") ؛
  اكتب_س(" ------------------------------------------------ ") ؛
  كرر
     س := ص؛     اكتب(س، " ") ؛
     طالما   س >=١٠٠٠   اعمل
       ابدأ  اكتب("m")؛   س := س - ١٠٠٠    نهاية ؛
     اذا  س >=٥٠٠  فان
       ابدأ  اكتب("d")؛   س := س - ٥٠٠    نهاية ؛
     طالما   س >= ١٠٠ اعمل
       ابدأ  اكتب("c")؛   س := س - ١٠٠    نهاية ؛
     اذا  س >=٥٠ فإن
       ابدأ  اكتب("l")؛   س := س - ٥٠    نهاية ؛
     طالما   س >=١٠   اعمل
       ابدأ  اكتب("x")؛   س := س - ١٠    نهاية ؛
     اذا  س >=٥  فإن
       ابدأ  اكتب("v")؛   س := س - ٥    نهاية ؛
     طالما   س >=١   اعمل
       ابدأ  اكتب("i")؛   س := س - ١    نهاية ؛
     اكتب_س؛   ص := ٢ * ص؛
  حتى ص > ٥٠٠٠
نهاية .
```



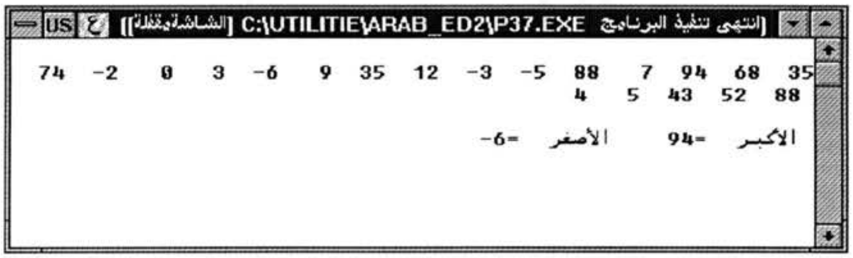
```
[انتهى تنفيذ البرنامج C:\UTILITIE\ARAB_ED2\P28.EXE [الشاشة مقفلة]]
الأرقام الرومانية هنا تقرأمن اليمين إلى اليسار
------------------------------------------------
   1            i
   2           ii
   4         iiii
   8         iiiv
  16          ivx
  32        iixxx
  64       iiiixl
 128      iiivxxc
 256        ivlcc
 512         iixd
1024      iiiixxm
2048   iiivxxxxmm
4096  ivxxxxlmmmm
```

7. Program 6.1 (Page 37)

Find the largest and smallest number in a given list.

This program mainly tests *while* and *if* statements, *array* of integers, and reading from a *file of integers.*

```
{برنامج ٦ . ١ صفحة ٣٧}
{ ايجاد أكبر وأصغر رقم في قائمة من الأرقام }

برنامج الأصغرو الأكبر (مدخل، مخرج)؛

ثابت  ن = ٢٠؛
متغير ل، ي ، ف ، الأصغر ، الأكبر : صحيح؛
    ق : مصفوفة [١ .. ن] من صحيح؛
    م : ملف من صحيح؛  { غير موجود في المثال}
ابدأ

{قراءة الأرقام، يفترض أن يكون عددها ٢٠، حسب المثال وسيتم
قراءتها من ملف }
 قراءة (م ،"minmax.inp") ؛
لقيم ل := ١ إلى ٢٠ اعمل      اقرأ (م ، ق[ل] ) ؛

{ الآن تبدأ عملية إيجاد الأصغر و الأكبر}

 الأصغر := ق[١] ؛  الأكبر := الأصغر ؛   ل := ٢ ؛
 طالما ل < ن      اعمل
 ابدأ ي := ق[ل] ؛ ف := ق[ل+١] ؛
     إذا  ي > ف فإن
     ابدأ  إذا  ي > الأكبر  فإن     الأكبر := ي؛
           إذا  ف < الأصغر  فإن     الأصغر := ف؛
     نهاية وإلا
     ابدأ  إذا  ف > الأكبر  فإن     الأكبر := ف؛
           إذا  ي < الأصغر  فإن     الأصغر := ي؛
     نهاية ؛
   ل := ل + ٢؛
 نهاية ؛
 إذا  ل = ن  فإن
     إذا  ق[ن] > الأكبر  فإن  الأكبر := ق[ن]
     وإلا  إذا  ق[ن] < الأصغر  فإن   الأصغر := ق[ن] ؛
 اكتب_س؛
 اكتب_س؛
 اكتب_س("الأكبر = "، الأكبر، "    الأصغر = "، الأصغر) ؛
نهاية .
```


[انتهى تنفيذ البرنامج C:\UTILITIE\ARAB_ED2\P37.EXE [الشاشة مقفلة]]
US
ع

```
 74  -2   0   3  -6   9  35  12  -3  -5  88   7  94  68  35
                                             4   5  43  52  88

                                     الأكبر =94     الأصغر =-6
```

8. Program 6.2 (Page 38)

The program plots a real-valued function *f(x)= exp(-x) * sin(2*pi*x)*. This program extends program 4.9 (page 30) to print *x-axis*. The *x-axis* is printed vertically. Due to the limitation of the output screen (not all the output will show in the output window, the result of the Arabic program is printed here directly from an output file that has been opened for this purpose. Note that the left hand side of the *x-axis* is the positive side (reverse of the English view).

This program mainly tests *for* statement, *sin* and *exp* functions, and *array* of characters.

```
{برنامج  ٦ . ٢ صفحة  ٣٨}
{رسم الدالة  د (س) = أس(-س) * جا (٢*ط*س)
مع رسم الاحداثي السيني (ولكن يرسم بشكل رأسي). هذا ابرنامج هو تطوير
لبرنامج  9.4 صفحة  ٣٠}

برنامج  رسم_دالة (مخرج)؛

ثابت  د = ٠,٠٦٢٥؛   { ١/١٦. ١٦ سطر للفترة [س،س+١]}
    ح = ٣١؛        {سعة الحروف في الفترة [ص،ص+١]}
    طول = ٦٧؛   { أقصى طول للسطر }
    م = ٣٤؛ { موقع المحور السيني }
    ط٢ = ٦,٢٨٣١٨؛  { ٢ * ط }
    حد = ٣٢؛

متغير س، ص : حقيقي؛   ع، ن : صحيح؛
    أ : مصفوفة [٠ .. طول] من حرف؛
    م : نص؛

ابدأ  كتابة (م، "sin.out")؛
    س := ٠؛
    لقيم ع:=٠  إلى  طول  اعمل  أ[ع] := " "؛
    لقيم ع:=١  إلى  حد  اعمل
    ابدأ  أ[م] := ":"؛
        ص := أس(-س) * جا (ط٢*س)؛
        ن := تقريب(ح*ص) + م؛
        أ[ن] := "*"؛
        اكتب_س(م، أ)؛
        أ[ن] := " "؛
        س := س + د؛
    نهاية
نهاية.
```

```
                        *
             *          :
     *                  :
*                       :
*                       :
   *                    :
         *              :
                *       :
                        *
                        :     *
                        :          *
                        :            *
                        :             *
                        :           *
                        :       *
                        :   *
                        *
                    *   :
                 *      :
               *        :
               *        :
                *       :
                  *     :
                     *  :
                        *
                        :*
                        :  *
                        :   *
                        :   *
                        :   *
                        : *
                        :*
```

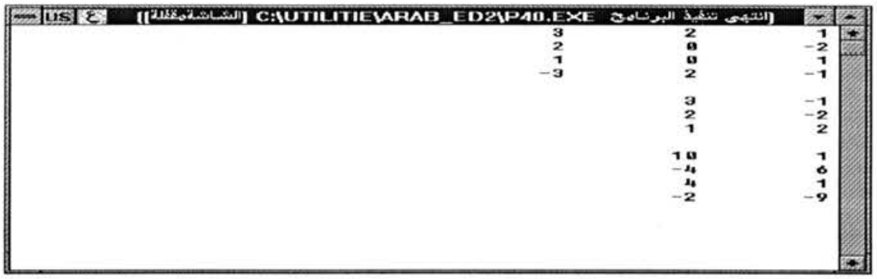
9. Program 6.3 (Page 40)

Matrix multiplication

This program mainly tests nested *for* statements and *two-dimensional array.*

```
{برنامج ٦ . ٣ صفحة ٤٠}
{ضرب المصفوفات}

برنامج ضرب_مصفوفات(مدخل، مخرج)؛

ثابت م= ٤ ؛ و = ٣؛ ن = ٢؛
متغير   ل : ١..م؛ ع : ١..ن؛ ك : ١..و؛
    س : صحيح؛
    أ : مصفوفة[١..م،١..و] من صحيح؛
    ب : مصفوفة[١..و،١..ن] من صحيح؛
    ج : مصفوفة[١..م،١..ن] من صحيح؛
    مل : ملف من صحيح؛    {يستخدم لقراءة الأعداد}
ابدأ
{اقرأ الأعداد، من ملف "matrix.inp"}
 قراءة(مل، "matrix.inp")
 لقيم ل:=١ إلى م اعمل
 ابدأ لقيم ك:=١ الى و اعمل
    ابدأ  اقرأ(مل،س)؛   اكتب(س)؛  أ[ل،ك] := س؛
    نهاية؛
```

```
  اكتب_س؛
 نـهايـة ؛
  اكتب_س؛

 لـقيـم   ك:=١  إلـى و  اعمل
  ابـد أ   لـقيـم  ع:=١  الـى ن  اعمل
          ابـد أ    اقـر أ (مل،س) ؛          اكتب(س) ؛   ب[ك، ع]  := س؛
          نـهايـة ؛
          اكتب_س؛
 نـهايـة ؛
  اكتب_س؛
{ اضرب    أ * ب  }
 لـقيـم   ل:=١  إلـى م  اعمل
  ابـد أ   لـقيـم  ع:=١  الـى ن  اعمل
          ابـد أ    س := ٠؛
                    لـقيـم   ك:=١  إلـى و  اعمل
                              س := س + أ[ل،ك] * ب[ك،ع] ؛
                    ج[ل،ع]  := س؛
                    اكتب(س)
          نـهايـة ؛
      اكتب_س
 نـهايـة ؛
  اكتب_س
نـهايـة .
```


US
ع
[انتهى تنفيذ البرنامج C:\UTILITIE\ARAB_ED2\P40.EXE [الشاشةمقفلة]]

```
3        2        1
2        0       -2
1        0        1
-3       2       -1

         3       -1
         2       -2
         1        2

        10        1
        -4        6
         4        1
        -2       -9
```

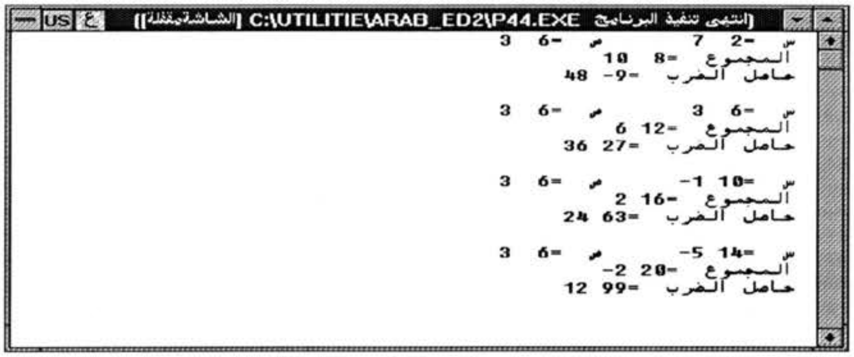
10. Program 7.1 (Page 44)

Operations (addition and multiplication) on complex numbers.

This program mainly tests usage of *records.*

```
{رنـامـج ٧ . ١ صفـحة  ٤٤}
{عملـيـات علـى الأعداد  الـمركبـة}

بـرنـامـج   عدد_مركب(مخرج) ؛
ثـابـت     عامل = ٤؛
نـوع    مركب = سجل
          ح، ت  :  صحيـح
       نـهايـة ؛
متـغيـر س،ص : مركب؛
       ن  : صحيـح؛
ابـد أ   س.ح := ٢؛      س.ت := ٧؛
       ص.ح := ٦؛      ص.ت := ٣؛
```

```
لقيم  ن :=  ١    إلى  ٤  اعمل
  ابدأ
     اكتب_س(" س = "، س.ح:٣، س.ت:٣، "     ص = "،
                ص.ح:٣، ص.ت:٣)؛
     {حساب  س + ص}
     اكتب_س(" المجموع = "، س.ح + ص.ح :٣،
                س.ت + ص.ت :٣)؛
     {حساب  س * ص}
     اكتب_س(" حاصل الضرب = "، س.ح*ص.ح - س.ت*ص.ت:٣،
                س.ح*ص.ت + س.ت*ص.ح :٣)؛
     اكتب_س؛
     س.ح := س.ح + عامل؛  س.ت := س.ت - عامل؛
  نهاية
نهاية.
```


[انتهى تنفيذ البرنامج C:\UTILITIE\ARAB_ED2\P44.EXE [الشاشةمقفلة]]
US
ع

```
س  2-  7        ص  6-  3
المجموع  8=  10
حاصل الضرب  -9-  48

س  6-  3        ص  6-  3
المجموع  12=  6
حاصل الضرب  27=  36

س  10=  -1        ص  6=  3
المجموع  16-  2
حاصل الضرب  63=  24

س  14=  -5        ص  6=  3
المجموع  20-  -2
حاصل الضرب  99=  12
```

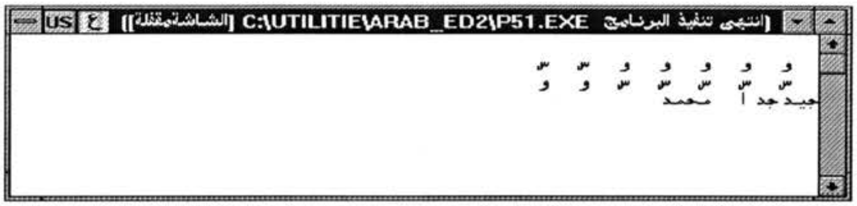
11. Program 8.1 (Page 51)

Example of set operations.

This program mainly tests *enumeration* and *set* types, operations on sets, and *procedure calls*.

```
{برنامج ٨ . ١، صفحة  ٥١}
  {مثال على المجموعات}

برنامج مجموعات(مخرج)؛
نوع   أيام = (اثنين، ثلاثاء، أربعاء، خميس، جمعة، سبت، أحد)؛
   أسبوع = مجموعة من أيام؛
متغير الأسبوع، عمل، عطلة : أسبوع؛
       يوم : أيام؛

عملية  فحص ( س : أسبوع)؛
 متغير       يوم : أيام؛
  ابدأ
     اكتب(" ")؛
     لقيم يوم:= اثنين  إلى  أحد  اعمل
      إذا  يوم في  س  فان  اكتب(" س")
      والا  اكتب(" و ")؛
     اكتب_س؛
```

```
نهاية ؛  { العملية فحص }

ابدأ  { البرنامج الرئيسي }
عمل := [] ؛  عطلة := [] ؛
{باسكال العربي حاليا لايقبل استخدام النطاق في المجموعات}
الأسبوع := [ اثنين،ثلاثاء، أربعاء،خميس،جمعة ،سبت، أحد ] ؛
يوم := سبت؛
عطلة := [يوم ] + عطلة + [ أحد ] ؛
فحص(عطلة ) ؛
عمل := الأسبوع - عطلة ؛
فحص(عمل ) ؛
إذا  عطلة <= الأسبوع  فإن  اكتب("جيد " ) ؛
إذا  الأسبوع >= عمل  فإن  اكتب("جدا " ) ؛
إذا ليس(عمل >= عطلة )  فإن اكتب("  محمد " ) ؛
{ الجملة الشرطية لاتقبل المجموعة المباشرة "[سبت]"  }
إذا عطلة <= عمل فإن  اكتب("  لاتهتم ! " ) ؛
اكتب_س
نهاية .
```



```
[انتهى تنفيذ البرنامج C:\UTILITIE\ARAB_ED2\P51.EXE [الشاشةمقفلة]]
```

```
و  و  و  و  و  س  س
س  س  س  س  س  و  و
جيدجدا  محمد
```

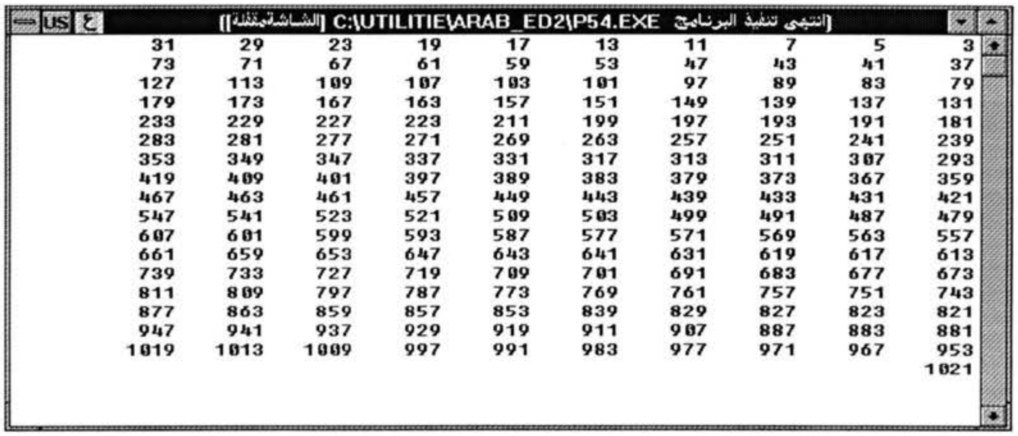
12. Program 8.2 (Page 54)

Generating prime numbers between 3 and around 1000. using a sieve containing odd integers in this range. The program has some implementation dependent constants (the maximum size of sets). To accommodate the prime numbers in one output screen, the prime numbers are computed up to 1000 instead of 10,000. The original program left the output undeveloped [JW74, p. 53], however, this program provides an output.

This program mainly tests *set* and *record* types, compound structures (*array* of sets), *with* statement, nested control statements, and *succ* function.

```
{برنامج ٨ . ٢ صفحة ٥٤}
{استخراج الأعداد الأولية الواقعة بين ٣ و ١٠٠٠ باستخدام فرز
 الأعداد الفردية الواقعة في هذا النطاق}

برنامج الأولية (مخرج) ؛
ثابت لم = ١٦؛      {أكبر عدد مسموح به في المجموعات}
    ل = ١٥؛       {لم - ١ }
    ض = ٣١؛       { ١٠٠٠ قاسم لم قاسم ٢ }
نوع    مج = مجموعة من ٠ .. ل ؛
```

```
متغير
   فرز ، أولي : مصفوفة [٠ .. ض] من مج ؛
   تالي : سجل   ك، ب : صحيح؛
        نهاية ؛
   ج، كك، ت، س : صحيح؛      لاشي : منطقي؛
   فارغة : مج؛
ابدأ
 فارغة := [ ]؛
 لقيم ت:= ٠  إلى  ض   اعمل
 ابدأ
   فرز[ت] := [٠،١،٢،٣،٤،٥،٦،٧،٨،٩،
            ١٠،١١،١٢،١٣،١٤،١٥]؛
   أولي[ت] := [ ]؛
 نهاية؛
 فرز[٠] := فرز[٠] - [٠]؛
 تالي.ك := ٠؛
 تالي.ب := ١؛
 لاشي := خطأ ؛

 مع تالي اعمل
  كرر  { جد العدد الأولي}
   طالما  ليس (ب في فرز[ك]) اعمل   ب := تابع(ب)؛
   أولي[ك] := أولي[ك] + [ب]؛
   س := ٢ * ب + ١؛
   ج := ب؛           كك := ك؛
   طالما  كك <= ض  اعمل   { استبعد غير الأولي }
   ابدأ
      فرز[كك] := فرز[كك] - [ج]؛
      كك := كك + ك*٢؛             ج := ج + س؛
      طالما  ج > ل  اعمل
      ابدأ  كك := كك + ١؛
            ج := ج - لم
      نهاية
   نهاية؛
   إذا  فرز[ك] = فارغة  فان
   ابدأ  لاشي := صواب؛  ب := ٠؛
   نهاية؛
   طالما  لاشي  و  (ك < ض)  اعمل
   ابدأ  ك := ك + ١؛
       لاشي :=   (فرز[ك] = فارغة)؛
   نهاية
  حتى  لاشي؛   { نهاية  مع}

 {طباعة الأعداد الأولية }

 ت := ١؛
 لقيم ج:=٠ إلى ض  اعمل
   لقيم س:=٠ إلى ل  اعمل
     اذا س فى أولي[ج]  فان
     ابدأ    اكتب(٢*ج*لم + ٢*س +١ :٧)؛
     ت := ت باقي ١٠ +١؛
         اذا (ت =١) فان اكتب_س؛
     نهاية
نهاية.
```


US
ع
[انتهى تنفيذ البرنامج C:\UTILITIE\ARAB_ED2\P54.EXE [الشاشةممتلئة]]

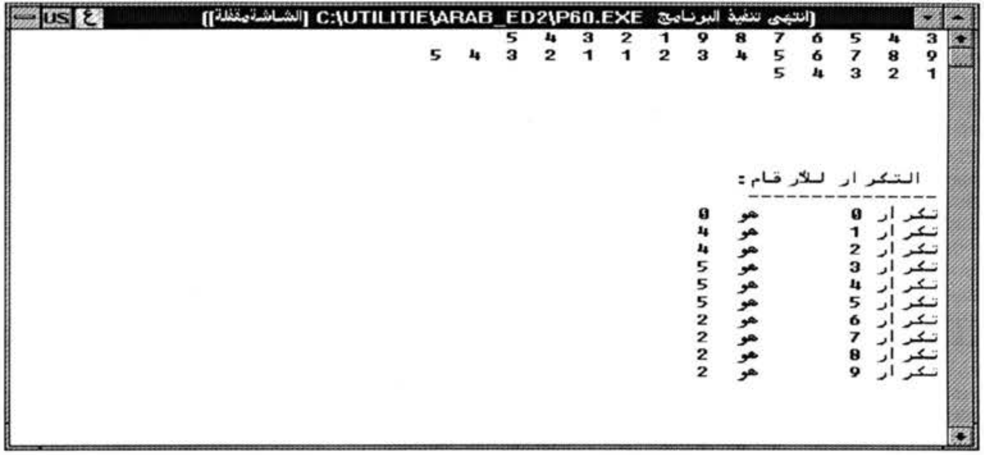
```
  31    29    23    19    17    13    11     7     5     3
  73    71    67    61    59    53    47    43    41    37
 127   113   109   107   103   101    97    89    83    79
 179   173   167   163   157   151   149   139   137   131
 233   229   227   223   211   199   197   193   191   181
 283   281   277   271   269   263   257   251   241   239
 353   349   347   337   331   317   313   311   307   293
 419   409   401   397   389   383   379   373   367   359
 467   463   461   457   449   443   439   433   431   421
 547   541   523   521   509   503   499   491   487   479
 607   601   599   593   587   577   571   569   563   557
 661   659   653   647   643   641   631   619   617   613
 739   733   727   719   709   701   691   683   677   673
 811   809   797   787   773   769   761   757   751   743
 877   863   859   857   853   839   829   827   823   821
 947   941   937   929   919   911   907   887   883   881
1019  1013  1009   997   991   983   977   971   967   953
                                                      1021
```

13. Program 9.1 (Page 60)

Frequency count of digits in an input file. The original program counts the frequency of letters, not digits. The Arabic Pascal has a limit of 16 elements in the set. Therefore, counting the frequency of alphabets is not applicable in the Arabic Pascal using sets.

This program mainly tests *array* and *set* types, and *eof* and *eoln* functions.

```
{برنامج ٩ . ١، صفحة ٦٠}
{حساب تكرار الأرقام في مجموعة من الأرقام. في البرنامج الأصل حساب تكرار
الحروففي مجموعة من الحروف، ولكن هذا غير ممكن في باسكال العربي لأن أقصى
عددللمجموعة هو ١٦ }

برنامج التكرار (مدخل، مخرج)؛

متغير رقم : حرف؛
    تكرار : مصفوفة["٠" .. "٩"] من صحيح؛
    أرقام : مجموعة من "٠" .. "٩"؛
    م : نص؛
ابدأ
    قراءة (م، "fcount.inp") ؛
    أرقام := ["٠"، "١"، "٢"، "٣"، "٤"، "٥"، "٦"، "٧"، "٨"، "٩"]؛
    لقيم رقم :="٠" إلى "٩" اعمل تكرار[رقم] :=٠؛
    طالما ليس نها_م (م) اعمل
    ابدأ
        طالما ليس نها_س(م) اعمل
        ابدأ اقرأ (م ،رقم) ؛ اكتب(رقم :٣) ؛
          اذا رقم في أرقام فان تكرار[رقم] := تكرار[رقم] + ١
        نهاية ؛
        اكتب_س؛      اقرأ_س(م)
    نهاية ؛
    اكتب_س؛
    اكتب_س؛
    اكتب_س("التكرار للأرقام :")؛
    اكتب_س("--------------");
    لقيم رقم :="٠" إلى "٩" اعمل
     اكتب_س("تكرار "، رقم:٥،"    هو    "،تكرار[رقم]:٥);

نهاية.
```


[انتهى تنفيذ البرنامج C:\UTILITIE\ARAB_ED2\P60.EXE [الشاشة مقفلة]]
US
ع

```
                  5   4   3   2   1   9   8   7   6   5   4   3
          5   4   3   2   1   1   2   3   4   5   6   7   8   9
                                              5   4   3   2   1

التكرار للارقام:
---------------
تكرار 0        هو   0
تكرار 1        هو   4
تكرار 2        هو   4
تكرار 3        هو   5
تكرار 4        هو   5
تكرار 5        هو   5
تكرار 6        هو   2
تكرار 7        هو   2
تكرار 8        هو   2
تكرار 9        هو   2
```

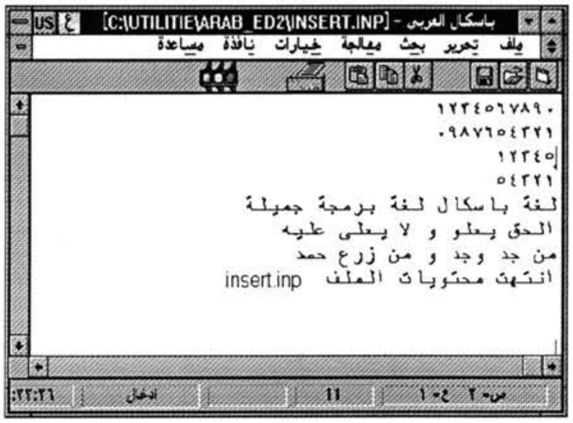
14. Program 9.2 (Page 61)

Inserting leading blanks at the beginning of each line while copying it from one file to another. The original program reads from the standard input and writes to the standard output, while this program reads from a file and writes to another file (more general, to test the file operations). Note that the execution window will not show up in running this program because there is no input nor output to the standard input/output. After the program source code, both input and output files are shown.

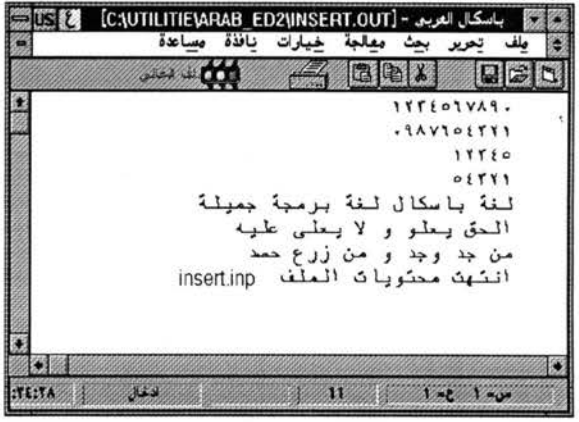
This program mainly tests reading from one file and writing to another file.

```
{ برنامج ٩ . ٢   صفحة ٦١}
{ نقل محتويات ملف إلى ملف آخر مع ادخال فراغات قبل كل سطر }
برنامج أدخل_فراغ;

ثابت  ن = ٢٠;
متغير    ح : حرف;
      م١، م٢ : نص;
ابدأ
      قراءة(م١، "insert.inp" );
      كتابة(م٢، "insert.out" );
      طالما ليس نهاية_م (م١)    اعمل
      ابدأ
          اكتب(م٢، "      ");
          طالما  ليس نهاية_س(م١)   اعمل
          ابدأ  اقرأ(م١، ح);   اكتب(م٢، ح)
```

```
        نـهايـة ؛
        اكـتـب_س(م ٢ ) ؛
     اقـرأ_س(م ١ ) ؛
  نـهايـة
نـهايـة.
```


[C:\UTILITIE\ARAB_ED2\INSERT.INP] - باسكال العربى
US
ع
ملف تحرير بحث معالجة خيارات نافذة مساعدة
س= ٢ ع= ١
11
ادخال
:٢٢:٢٦

```
١٢٣٤٥٦٧٨٩٠
٠٩٨٧٦٥٤٣٢١
١٢٣٤٥
٥٤٣٢١
لغة باسكال لغة برمجة جميلة
الحق يعلو و لا يعلى عليه
من جد وجد و من زرع حصد
انتهت محتويات الملف   insert.inp
```


[C:\UTILITIE\ARAB_ED2\INSERT.OUT] - باسكال العربى
US
ع
ملف تحرير بحث معالجة خيارات نافذة مساعدة
س= ١ ع= ١
11
ادخال
:٢٤:٢٨

```
١٢٣٤٥٦٧٨٩٠
٠٩٨٧٦٥٤٣٢١
١٢٣٤٥
٥٤٣٢١
لغة باسكال لغة برمجة جميلة
الحق يعلو و لا يعلى عليه
من جد وجد و من زرع حصد
انتهت محتويات الملف   insert.inp
```

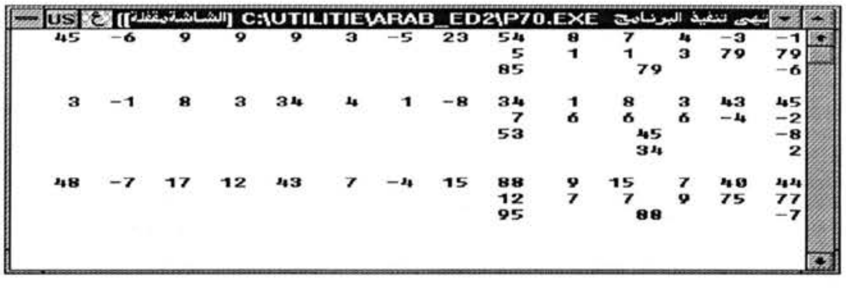
15. Program 11.2 (Page 70)

Find the minimum, the maximum, and the difference between the minimum and the maximum for a list of numbers. This program is an extension to program 11.1.

This program mainly tests *array* type and *call-by-reference*.

```
{برنامج  ١١ . ٢  صفحة  ٧٠}
{ ايجادالأكبر والآصغر في قائمة من الأرقام وايجاد الفرق بين الأكبر والأصغر
 هذا البرنامج هو تطوير للبرنامج  ١١ . ١  صفحة ٦٨ }

برنامج  الأكبر_والأصغر (مخرج) ؛
ثابت  ن = ٢٠؛
نوع   قائمة = مصفوفة[١..ن] من صحيح؛
متغير   أ،ب : قائمة ؛
   ع، ل، الأصغر١ ، الأكبر١، الأصغر٢ ، الأكبر٢ : صحيح؛
   م : ملف من صحيح؛

عملية  أكبروأصغر (متغير ق:قائمة ؛ متغير  ج،ك:صحيح) ؛
{كلمة  "متغير" الأولى ليس لها  داع لأن قسمة  ق  لاتتغير }
متغير    ع : ١..ن؛
   ي، ف : صحيح؛
ابدأ   ج := ق[١] ؛   ك := ج؛          ع := ٢؛
   طالما   ع<ن   اعمل
   ابدأ   ي := ق[ع] ؛   ف := ق[ع+١] ؛
      اذا ي > ف  فإن
   ابدأ   إذا   ي > ك  فإن   ك := ي؛
          إذا   ف < ج  فإن   ج := ف؛
   نهاية و إلا
   ابدأ   إذا   ف > ك  فإن   ك := ف؛
```

```
          إذا ي < ج فإن ج := ي؛
        نهاية ؛
      ع := ع + ٢؛
    نهاية ؛
    إذا ع = ن فإن
      إذا ق[ن] > ك فإن ك := ق[ن]
      وإلا إذا ق[ن] < ج فإن ج := ق[ن]؛
نهاية ؛ { أصغرو أكبر }

ابدأ
  قراءة (م ، "maxmin.inp" ) ؛
  لقيم ع := ١ إلى ٢٠ اعمل ابدأ   اقرأ (م ، أ[ع]) ؛ اكتب(أ[ع]:٤) ؛ نهاية ؛
  اكتب_س؛
  أكبرو أصغر (أ، الأصغر١، الأكبر١) ؛
  اكتب_س(الأصغر١، الأكبر١، الأكبر١ - الأصغر١) ؛ اكتب_س؛

  لقيم ع := ١ إلى ٢٠ اعمل ابدأ   اقرأ (م ، ب[ع]) ؛ اكتب(ب[ع]:٤) ؛ نهاية ؛
  اكتب_س؛
  أكبرو أصغر (ب، الأصغر٢، الأكبر٢) ؛
  اكتب_س(الأصغر٢، الأكبر٢، الأكبر٢ - الأصغر٢) ؛
  اكتب_س(مطلق (الأصغر١-الأصغر٢) ، مطلق (الأكبر١-الأكبر٢) ) ؛ اكتب_س؛

  لقيم ع:=١ إلى ن اعمل
    ابدأ أ[ع] := أ[ع] + ب[ع]؛
      اكتب(أ[ع]:٤) ؛
    نهاية ؛
  اكتب_س؛
  أكبرو أصغر (أ، الأصغر١، الأكبر١) ؛
  اكتب_س(الأصغر١، الأكبر١، الأكبر١ - الأصغر١) ؛
نهاية .
```



```
 45   -6    9    9    9    3   -5   23   54    8    7    4   -3   -1
                                          5    1    1    3   79   79
                                         85        79             -6

  3   -1    8    3   34    4    1   -8   34    1    8    3   43   45
                                          7    6    6    6   -4   -2
                                         53        45             -8
                                                   34              2

 48   -7   17   12   43    7   -4   15   88    9   15    7   40   44
                                         12    7    7    9   75   77
                                         95        88             -7
```

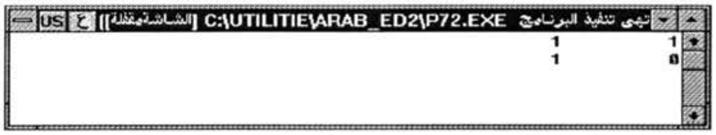
16. Program 11.3 (Page 72)

Procedure parameters.

This program mainly tests the difference between *call-by-value* and *call-by-reference parameter passing conventions*.

```
{برنامج ١١ . ٣، صفحة ٧٢}
{مثال على استخدام عوامل العمليات}
```

```
برنامج   عوامل (مخرج) ؛

متغير   أ، ب : صحيح ؛

عملية   ع(س : صحيح؛ متغير ص: صحيح) ؛
ابدأ
    س := س + ١ ؛
    ص := ص + ١ ؛
    اكتب_س(س، ص) ؛
نهاية ؛

ابدأ
    أ := ٠ ؛
    ب := ٠ ؛
    ع ( أ، ب) ؛
    اكتب_س(أ، ب) ؛
نهاية .
```



تهى تنفيذ البرنامج C:\UTILITIE\ARAB_ED2\P72.EXE [الشاشة مقفلة]

```
1          1
1          0
```

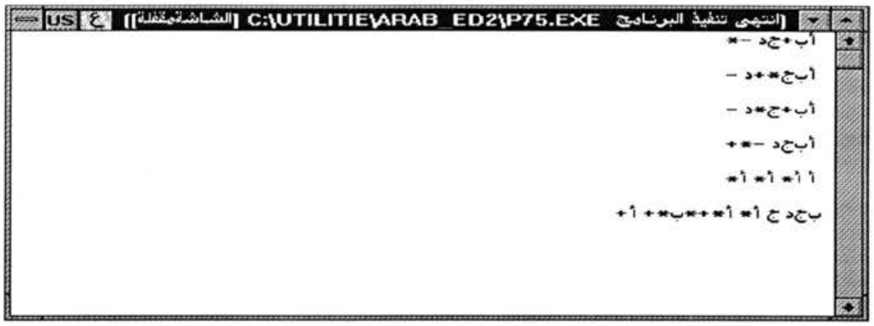
17. Program 11.4 (Page 75)

Construct a program to convert the infix expression into postfix form. The program uses the recursive approach instead of the stack implementation.

This program mainly tests *nested* procedures, *recursive* procedures.

```
{برنامج ١١ . ٤ صفحة   ٧٥}
{تحويل العبارة الرياضية إلى عمليات لاحقة. مثال
"س + ص" تتحول إلى "س ص+ "
المدخلات (كما في الكتاب صفحة ٧٣) وهي كالتالي:
(أ+ب) * (ج-د )
أ+ب*ج-د
(   أ    +  ب) *        ج-د
أ+ب* (ج-د )
أ • أ • أ • أ
ب+ج* (د +ج* أ * أ ) *ب+أ .
}
برنامج   عملية_لاحقة (مدخل ،مخرج) ؛

متغير   ح : حرف؛
        م : نص؛

عملية جد ؛
ابدأ  كرر   اقرأ (م ،ح)
      حتى (ح <> " " )    { يمكن الاستغناء عن الشرط الثاني }
نهاية ؛
```

```
عملية عبارة؛
متغير ع : حرف؛

    عملية حد؛

        عملية عامل؛
        ابدأ اذا ح = "(" فان
            ابدأ جد؛ عبارة؛ { ح(=) }
            نهاية
          والا اكتب(ح)؛
          جد
        نهاية؛ { عامل }
    ابدأ عامل؛
        طالما ح = "*" اعمل
        ابدا جد؛ عامل؛ اكتب("*")؛
        نهاية
    نهاية؛ { حد }
ابدا حد؛
    طالما (ح = "+") أو (ح = "-") اعمل
    ابدأ ع := ح؛ جد؛ حد؛ اكتب(ع)؛
    نهاية
نهاية؛ { عبارة }

ابدأ قراءة(م، "post.inp")؛
    جد؛
    كرر اكتب(" ")؛
        عبارة؛
        اكتب_س؛
        اكتب_س؛
    حتى ح = "."
نهاية.
```



[انتهى تنفيذ البرنامج C:\UTILITIE\ARAB_ED2\P75.EXE [الشاشة ممتلئة]]
ع
US

```
أب+جد-*
أبج*+د-
أب+ج*د-
أبجد-*+
أأ*أ*أ*
بجدج أ*أ*+*ب*++أ+
```

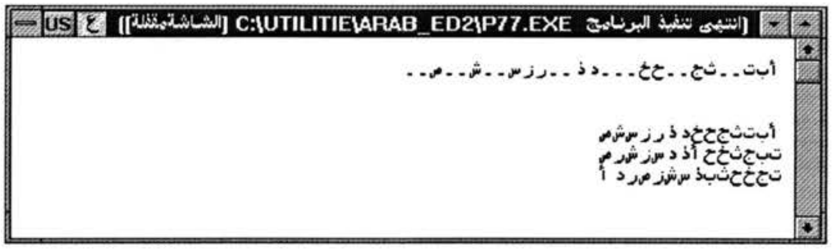
18. Program 11.5 (Page 77)

Binary tree traversal. The original program uses unusual procedure names; it uses *postorder* to mean *inorder* and *endorder* for *postorder*.

This program mainly tests *pointer* and *record* types, *call-by-reference*, recursive calls, and function calls (*enter* procedure has been converted to a function to test function calls).

```
{برنامج ١١ . ٥ صفحة ٧٧}
{استعراض عناصر شجرة ثنائية}

برنامج استعراض(مدخل، مخرج)؛

نوع مؤشر = ^عنصر؛
   عنصر = سجل
     س : حرف؛
     يمين، يسار : مؤشر؛
  نهاية؛
متغير جذر : مؤشر؛
     ح : حرف؛
     م : نص؛

عملية قبلية(ش : مؤشر)؛
ابدأ اذا ش <> فارغ فان
     ابدأ اكتب(ش^.س)؛
          قبلية (ش^.يسار)؛
          قبلية (ش^.يمين)؛
     نهاية؛
نهاية؛

عملية وسطية(ش : مؤشر)؛
ابدأ اذا ش <> فارغ فان
     ابدأ
          وسطية (ش^.يسار)؛
          اكتب(ش^.س)؛
          وسطية (ش^.يمين)؛
     نهاية؛
نهاية؛

عملية بعدية(ش : مؤشر)؛
ابدأ اذا ش <> فارغ فان
     ابدأ
          بعدية (ش^.يسار)؛
          بعدية (ش^.يمين)؛
          اكتب(ش^.س)؛
     نهاية؛
نهاية؛

دالة أدخل: مؤشر؛
متغير ش : مؤشر؛
ابدأ اقرأ(م ،ح)؛ اكتب(ح)؛
     اذا (ح <> ".") فان
     ابدأ جديد (ش)؛
          ش^.س := ح؛
          ش^.يسار := أدخل؛
          ش^.يمين := أدخل؛
     نهاية
     والا ش := فارغ؛
```

```
    أدخل := ش؛
نهاية ؛

ابدأ   { البرنامج الرنيسي }

    قراءة (م ، "tree.inp") ؛

    اكتب(" ") ؛  جذر := أدخل؛ اكتب_س؛
    اكتب_س؛ اكتب_س؛
    اكتب(" ") ؛  قبلية (جذر ) ؛  اكتب_س؛
    اكتب(" ") ؛  وسطية (جذر ) ؛  اكتب_س؛
    اكتب(" ") ؛  بعدية (جذر ) ؛  اكتب_س؛
نهاية .
```


[انتهى تنفيذ البرنامج C:\UTILITIE\ARAB_ED2\P77.EXE [الشاشةممثلة]]
US
ع

```
أبت..ثج..حخ...دذ..رزس..ش..ص..

أبتثجحخد ذ ر ز سشص
تبجثخح أذ د سزشرص
تجخحثبذ سشزصرد أ
```

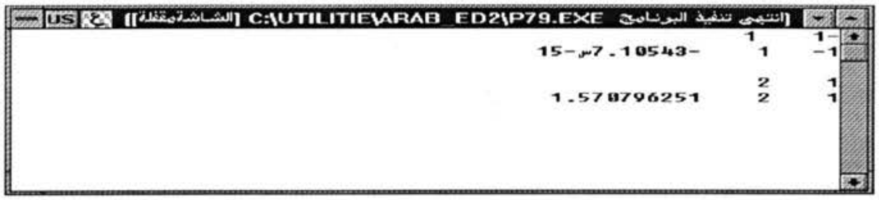
19. Program 11.6 (Page 79)

Find zero of a function by bisection. The original program caused an infinite loop in the second input (for evaluating the *cos* function). I have tested this program using Turbo Pascal and the same thing happens. It seems that the problem comes from the limitation of the real numbers. Therefore, I have added a counter to stop the looping after 100,000 iterations. The APC does not accept a function as a parameter. So, the function parameter in the original program has been transformed to a boolean parameter that activates the required function.

This program mainly tests function calls with parameters.

```
{برنامج  ١١ . ٦ صفحة  ٧٩}
{ ايجاد صفر دالة باستخدام التقسيم الثناني}

برنامج تقسيم_ثناني(مدخل، مخرج) ؛
ثابت    ابسلون = ١س-١٤؛
متغير س، ص : حقيقي؛

دالة  صفر (شفره: منطقي؛ أ،ب : حقيقي) : حقيقي؛
{باسكال العربي لايقبل وجود اسم دالة داخل العوامل}
```

```
متغير    س، ع : حقيقي؛    ج : منطقي؛ عدد : حقيقي؛
ابدأ    اذا شفره    فان    ج := جا (أ) < ٠
       والا ج:= جتا (أ)< ٠؛
       عدد := ١؛
       كرر
           عدد := عدد +١؛
           س := (أ+ب)/ ٢,٠؛
           اذا شفره  فان  ع := جا (س)
           والا ع:= جتا (س)؛
           إذا  (ع<٠)= ج  فان  أ := س
           والا  ب := س؛
       حتى   (مطلق(أ-ب) < ابسلون) أو (عدد > ١٠٠٠٠٠,٠)؛
       صفر := س؛
نهاية؛   { صفر }

ابدأ   { الرنيسي }
 اقرأ(س، ص)؛    اكتب_س(س،ص، صفر (صواب،س،ص))؛
 اقرأ(س، ص)؛    اكتب_س(س،ص، صفر (خطأ ،س،ص))؛
نهاية.
```


[انتهى تنفيذ البرنامج C:\UTILITIE\ARAB_ED2\P79.EXE [الشاشة مقفلة]]
US

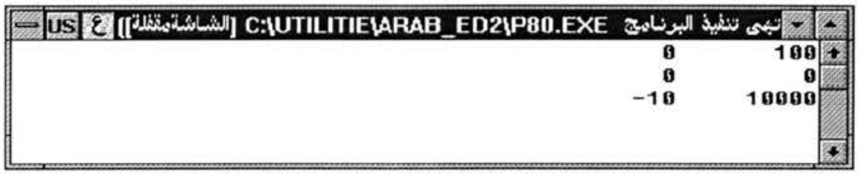
```
                         1       1-
  15-س7.10543-           1      -1

                         2       1
  1.570796251            2       1
```

20. Program 11.7 (Page 80)

Testing the side effect that may be caused by using functions.

This program mainly tests function calls.

```
{برنامج  ١١ . ٧  صفحة ٨٠}
{اختبار الأثر الجانبي لاستخدام الدوال}

برنامج   أثر_جانبي(مخرج)؛
متغير      أ، ز : صحيح؛

دالة  ت(س: صحيح): صحيح؛
ابدأ    ز := ز - س؛  { تأثير جانبي على ز }
        ت := مربع(س)؛
نهاية؛

ابدأ
      ز := ١٠؛  أ := ت(ز)؛  اكتب_س(أ،ز)؛
      ز := ١٠؛  أ := ت(١٠) * ت(ز)؛  اكتب_س(أ،ز)؛
      ز := ١٠؛  أ := ت(ز) * ت(١٠)؛  اكتب_س(أ،ز)؛
نهاية.
```


تهى تنفيذ البرنامج C:\UTILITIE\ARAB_ED2\P80.EXE [الشاشة مقفلة]
US
ع

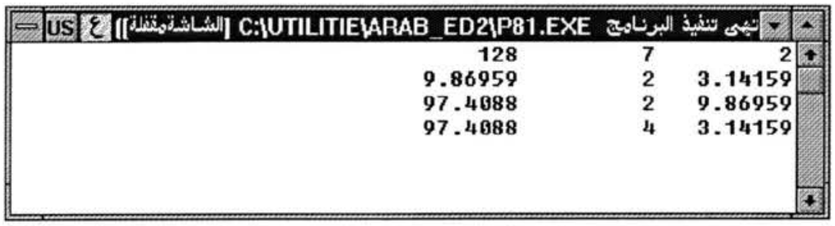
```
                0        100
                0          0
              -10      10000
```

21. Program 11.8 (Page 81)

Compute the value of *x* to the power of *y*. This program is an extension to program 4.8 in page 29.

This program mainly tests function calls.

```
{برنامج ١١ . ٨ صفحة ٨١}
{ايجاد الأس لعددين. هذا البرنامج هو تطوير للبرنامج ٨ . ٤ صفحة ٢٩}

برنامج الأس(مخرج)؛
متغير  ط، مط : حقيقي؛

دالة قوة(س:حقيقي؛ ص:صحيح) : حقيقي؛   {ص>=٠}
 متغير   ع : حقيقي؛
 ابدأ  ع := ١؛
    طالما ص>٠  اعمل
    ابدأ  طالما ليس فردي(ص)  اعمل
          ابدأ  ص := ص قاسم ٢؛
                س := مربع(س)؛
          نهاية؛
          ص := ص - ١؛    ع := س * ع
    نهاية؛
    قوة := ع؛
نهاية؛  {قوة}

ابدأ  ط := ٣,١٤١٥٩؛
    اكتب_س(٢,٠، ٧ ، قوة(٢,٠،٧))؛
    مط := قوة(ط، ٢)؛
    اكتب_س(ط، ٢، مط)؛
    اكتب_س(مط، ٢ ، قوة(مط، ٢))؛
    اكتب_س(ط، ٤، قوة(ط، ٤))
نهاية.
```


تهى تنفيذ البرنامج C:\UTILITIE\ARAB_ED2\P81.EXE [الشاشة مقفلة]
US
ع

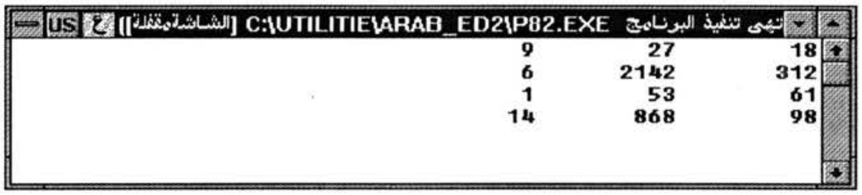
```
      128        7          2
  9.86959        2    3.14159
  97.4088        2    9.86959
  97.4088        4    3.14159
```

22. Program 11.9 (Page 82)

Recursive formulation of *gcd*.

This program mainly tests recursive function calls and procedure calls.

```
{برنامج ١١ . ٩ صفحة ٨٢}
{ ايجاد القاسم المشترك الأكبر باستخدام العملية المتعاودة}

برنامج   القاسم_المشترك_الأكبر (مخرج) ؛
متغير     س، ص، ن : صحيح؛

دالة    القاسم (م ،ن :صحيح) :صحيح ؛
ابدأ
    اذا  ن=٠  فان  القاسم  := م
    والا   القاسم  := القاسم (ن، م  باقي ن) ؛
نهاية ؛  { القاسم }

عملية   جرب  (أ ،ب : صحيح) ؛
ابدا     اكتب_س(أ ،ب، القاسم (أ ،ب))
نهاية ؛

ابدا
    جرب(١٨، ٢٧) ؛
    جرب(٣١٢، ٢١٤٢) ؛
    جرب(٦١، ٥٣) ؛
    جرب(٩٨، ٨٦٨)
نهاية .
```



[الشاشةمقفلة] C:\UTILITIE\ARAB_ED2\P82.EXE تهى تنفيذ البرنامج

```
        9        27        18
        6      2142       312
        1        53        61
       14       868        98
```

23. Complementary Example

This example is not in Jensen and Wirth book. It contains some concepts that are not mentioned in that book, namely:

1. *Case* statement.

2. Built-in procedures: *new* and *dispose* (dynamic allocation and deallocation).

3. Mutual recursion. Two procedures, each one calls the other, communicating through a variable parameter.

4. *Call-by-reference* utilizing a non simple type (array type).

5. *Goto* statement.

6. Built-in functions: *arctan, ord, chr, ln, pred,* and *sqrt.*

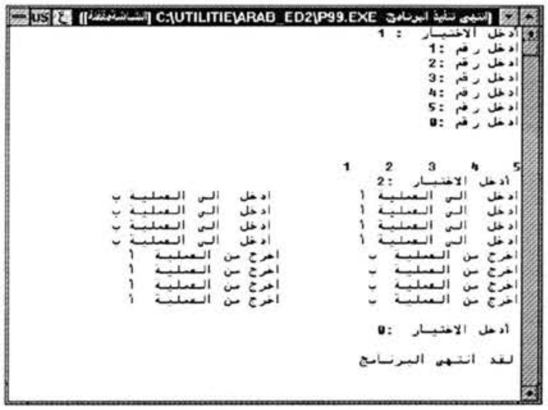
The full translation of this example is available at the end of this example.

```
{ هذا البرنامج مكمل للبرامج السابقة، حيث يشتمل على بعض
الخواص
التي لم يرد ذكرها في البرامج السابقة.
   ١. حالة     ٢. اذهب_الى     ٣. لاحق
   ٤. الدوال التي لم يرد ذكرها في السابق
   ٥. عمليتا جديد و تخلص
}

برنامج    تكميل(مدخل، مخرج)؛

عنوان    ٩٩؛
نوع قائمة = مصفوفة[١..٢٠٠] من صحيح؛
متغير
 اختيار، ع، طول : صحيح؛
 مواصلة : منطقي؛
 ق : قائمة؛

عملية مصفوفة_ستاك؛
{ انشاء قائمة باستخدام المؤشر ثم عرضها كما لو كانت في مصفوفة
الستاك }
نوع
   قاطرة = ^رقم؛
   رقم = سجل
        ن : صحيح؛
        بعد : قاطرة؛
   نهاية؛
متغير
   عنصر ، مؤشر : قاطرة؛

 عملية القراءة( جد : قاطرة)؛
 ابدا
   اكتب("ادخل رقم :")؛
   اقرا (جد^.ن)؛
 نهاية؛

ابدأ {مصفوفة_ستاك}
   جديد (عنصر)؛
   عنصر^.بعد := فارغ؛
   القراءة(عنصر)؛
   مؤشر := عنصر؛
   طالما (عنصر^.ن <> ٠)    اعمل
```

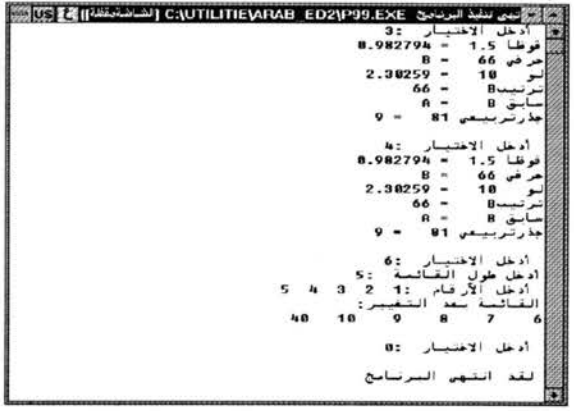
```
    ابدأ
        جديد (عنصر ) ؛
        القراءة (عنصر ) ؛
        عنصر^.بعد :=مؤشر ؛
        مؤشر :=عنصر
    نهاية ؛
    مؤشر :=عنصر^.بعد ؛

{ اكتب السلسلة بشكل عكسي بدون كتابة الصفر }
    اكتب_س؛
    اكتب_س؛
    طالما  (مؤشر <> فارغ)  اعمل
    ابدأ
        عنصر  := مؤشر ؛
        اكتب(عنصر^.ن:٥) ؛
        مؤشر  := عنصر^.بعد ؛
        تخلص(عنصر ) ؛
    نهاية ؛
نهاية ؛

عملية  استدعاء_متبادل؛
ثابت    أعلىمستوى = ٣؛
متغير   مستوى : صحيح؛

عملية   ب(متغير  ك  : صحيح ) ؛ لاحق؛
عملية   أ (متغير  ك  : صحيح ) ؛
ابدأ
    اكتب("ادخل  الى العملية أ") ؛
    ب(ك) ؛
    اكتب_س("       اخرج من العملية  أ") ؛
نهاية ؛

عملية   ب؛
ابدأ
    اكتب_س("      ادخل  الى العملية ب") ؛
    ك :=  ك + ١ ؛
    اذا  ك <= أعلىمستوى  فان    أ(ك) ؛
    اكتب("اخرج من العملية  ب") ؛
نهاية ؛
ابدأ  { عملية   استدعاء_متبادل }
  مستوى  := ٠ ؛
  أ (مستوى) ؛
نهاية ؛

عملية  دوال؛
{تجريب بعض الدوال في لغة باسكال العربية }
```

```
متغير ن : حقيقي؛
  ح : حرف؛
  ص : صحيح؛
ابدأ
  ن := ١,٥؛  اكتب_س("قوطا "، ن:٤ : ٢، " = "، قوطا (ن))؛
  ص := ٦٦؛  اكتب_س("حرفي "، ص:٤، " = "، حرفي(ص))؛
  ص := ١٠؛  اكتب_س("لو "، ص:٤، " = "، لو (ص))؛
  ح := "B"؛  اكتب_س("ترتيب"، ح:٤، " = "، ترتيب(ح))؛
             اكتب_س("سابق "، ح:٤، " = "، سابق(ح))؛
  ص := ٨١؛  اكتب_س("جذرتربيعي "، ص:٤، " = "، جذر_٢(ص))؛
نهاية؛

عملية مصفوفة_متغيرة( متغير س: قائمة؛متغير عدد :صحيح)؛
متغير مج، ع : صحيح؛
ابدأ { أضف على كل عنصر دليله، ثم ضع المجموع في الأخير }
  مج := ٠؛
  لقيم ع:=١ الى عدد اعمل
  ابدأ
    ق[ع] := ق[ع] + عدد؛
    مج := مج + ق[ع]؛
  نهاية؛
  عدد := عدد + ١؛
  ق[عدد] := مج؛
نهاية؛

ابدأ { البرنامج الرئيسي }
  مواصلة := صواب؛
  طالما مواصلة اعمل
  ابدأ
    اكتب("أدخل الاختيار : ")؛
    اقرا( اختيار)؛
    حالة اختيار من
      ١: مصفوفة_ستاك؛
      ٢: استدعاء_متبادل؛
      ٣،٤،٥ : دوال؛
      ٦: ابدأ
        اكتب("أدخل طول القائمة : ")؛
        اقرا(طول)؛
        اكتب("أدخل الأرقام : ")؛
        لقيم ع:=١ الى طول اعمل
        ابدأ
          اقرأ( ق[ع])؛
        نهاية؛
        اكتب_س("القائمة بعد التغيير:")؛
        مصفوفة_متغيرة(ق، طول)؛
        لقيم ع:=١ الى طول اعمل
          اكتب(ق[ع]:٥)؛
```

```
                  اكتب_س؛
                 نهاية ؛
        ٠ : اذهب_الى   ٩٩؛
         نهاية ؛  { حالة }
                  اكتب_س؛
        نهاية  ؛  { طالما }

٩٩  :  اكتب_س؛
  اكتب_س(" لقد انتهى البرنامج ")؛
نهاية.
```



```
انتهى تنفيذ البرنامج C:\UTILITIE\ARAB_ED2\P99.EXE [الشاشةمنقفلة]
أدخل الاختيار : 1
ادخل رقم :1
ادخل رقم :2
ادخل رقم :3
ادخل رقم :4
ادخل رقم :5
ادخل رقم :0

1   2   3   4   5
أدخل الاختيار :2
ادخل  الى العملية أ          ادخل  الى العمليه ب
ادخل  الى العملية أ          ادخل  الى العملية ب
ادخل  الى العملية أ          ادخل  الى العملية ب
ادخل  الى العملية أ          ادخل  الى العملية ب
اخرج من العملية  ب           اخرج من العملية  أ
اخرج من العملية  ب           اخرج من العملية  أ
اخرج من العملية  ب           اخرج من العملية  أ
اخرج من العملية  ب           اخرج من العملية  أ

أدخل الاختيار :0

لقد انتهى البرنامج
```



```
انتهى تنفيذ البرنامج C:\UTILITIE\ARAB_ED2\P99.EXE [الشاشةمنقفلة]
أدخل الاختيار :3
قوطا 1.5 = 0.982794
حرفي 66 = B
لو  10 = 2.30259
ترتيبB = 66
سابق B = A
جذرتربيعي 81 = 9

أدخل الاختيار :4
قوطا 1.5 = 0.982794
حرفي 66 = B
لو  10 = 2.30259
ترتيبB = 66
سابق B = A
جذرتربيعي 81 = 9

أدخل الاختيار :6
ادخل طول القائمة :5
أدخل الأرقام :1  2  3  4  5
القائمة بعد التغيير:
6   7   8   9   10   40

أدخل الاختيار :0

لقد انتهى البرنامج
```

```
{This program complements the previous examples.}

program complement(input, output);

label  99;
type    list = array[1..20] of integer;
var
   choice, count, len : integer;
   cont : boolean;
   l : list;
procedure  Stack_array;
{ creats and then displays a last-in first-out linked list
  it tests the procedures NEW and DISPOSE }
type
   link = ^num;
   num = record
           n : integer;
           next : link;
   end;
var
   p1, p2 : link;

   procedure  reading(new : link);
   begin
       write('Enter a number:');
       read(new^.n);
   end;
begin  { stack_array }
   new(p1);
   p1^.next := nil;
   reading(p1);
   p2 := p1;
   while (p1^.n <> 0)  do
   begin
        new(p1);
        reading(p1);
        p1^.next := p2;
        p2 := p1;
   end;
   p2 := p1^.next;
   { write the list in reverse order }
   writeln;
   writeln;
   while (p2 <> nil)  do
   begin
      p1 := p2;
      write(p1^.n : 5);
      p2 := p1^.next;
      dispose(p1);
   end;
end;

procedure mutualRec;
 { testing mutual recursion }
 const  maxlevel = 3;
 var    level : integer;

 procedure  B(var  k : integer); forward;
 procedure  A(var  k : integer);
 begin
      write('Enter procedure  A');
      B(k);
      writeln('       exit from procedure  A');
 end;

 procedure  B;
 begin
      writeln('     Enter procedure   B');
      k := k + 1;
      if k <= maxlevel   then   A(k);
      write('exit from procedure  B');
 end;
begin   { mutualRec  }
    level := 0;
    A(level);
end;    { mutualRec }
```

```
procedure functions;
  { test some of the built-in functions }
  var   n : real;
        c : char;
        i : integer;
  begin
       n := 1.5;  writeln('arctan ', n:4:2, ' = ',
arctan(n));
       i := 66;   writeln('chr ', i : 4, ' = ', chr(i));
       i := 10;   writeln('ln ', i : 4, ' = ', ln(i));
       c := 'B';  writeln(' ord ', c : 4, ' = ', ord(c));
                  writeln(' pred ', c : 4, ' = ', pred(c));
       i := 81;   writeln(' sqrt ', i: 4, ' = ', sqrt(i));
  end;

procedure  varArray(var  x : list;  var  no: integer);
{ add to each element in the array the list length,
  then put the total of all elemtns at the end of the list
(index no+1)
  and return the array}
   var   sum, k : integer;
   begin
        sum := 0;
        for k := 1  to  no  do
        begin
             x[k] := x[k] + no;
             sum := sum + x[k];
        end;
        no := no + 1;
        x[no] := sum;
   end;

begin    { main program }

   cont := true;
   while  cont  do
   begin
        writeln;
        write('Enter the choice :');
        read(choice);
        case  choice  of
          1 : stack_array;
          2 : mutualRec;
          3,4,5 : functions;
          6 : begin
                write('Enter the list lenght :');
                read(len);
                write('Enter the numbers :');
                for count:= 1  to  len do
                begin   read(l[count]);
                end;
                writeln('The list after the change :');
                varArray(l, len);
                for count:=1  to len  do
                    write(l[count]:5);
                writeln;
              end;
          0 : GOTO  99;
        end;   {case}
   end;  { While }

   99 : writeln;
        writeln('End of program');
end.
```

2

VITA

AbdulMalik S. Al-Salman

Candidate for the Degree of

Doctor of Philosophy

Thesis: AN ARABIC PROGRAMMING ENVIRONMENT

Major Field: Computer Science

Biographical:

Personal Data: Born in Riyadh, Saudi Arabia, On October 16, 1966.

Education: Graduated from El-Shafa High School, Riyadh, Saudi Arabia, in June 1983; received Bachelor of Science degree in Computer Science from King Saud University, Riyadh, Saudi Arabia in June 1988; received Master of Science degree in Computer Science from University of Georgia, Athens, Georgia in July 1992.
Completed the requirements for the Doctor of Philosophy with a major in Computer Science at Oklahoma State University in July 1996.

Experience: employed as systems analyst by the Royal Saudi Marine Forces in Summer 1986 and 1987; employed by King Saud University, Department of Computer Science as teaching assistant in 1988-1989.

Professional Memberships: Association for Computing Machinery (ACM).