1985

SCOUT| An automated football scouting system

K. Garry Dyer
The University of Montana

SCOUT

An Automated Football Scouting System

by

K. Garry Dyer

B.A., University of Montana, 1975

Presented in Partial Fulfillment of the Requirements for the Degree of

Master of Science

UNIVERSITY OF MONTANA

1985

Approved by:

[illegible]
Chairman, Board of Examiners

[illegible]
Dean; Graduate School

May 3, 1985
Date

Dyer, K. Garry, M.S., March 14, 1985 Computer Science

Scout: An Automated Football Scouting System (235 pp.)

Director: Alden Wright AHW

Football scouting is done in many ways. This paper explains how a computerized scouting system was analyzed and designed. It contains sections which discuss each of these phases of the development of the system along with a discussion of the implementation of the system. The system was made for the University of Montana football team, written in PASCAL and implemented on the University of Montana's DEC-2060 computer.

ACKNOWLEDGEMENTS

I would like to thank the following members of my committee for their help with the writing of this project: Dr. Alden Wright, Dr. Kenneth Silvestro, and Dr. Keith Yale. I also want to thank Dr. John Barr for his help while he was on my committee. A special thank you goes to Gary Frazer for his assistance with proofreading. Last but certainly not least I want to thank my wife Shirley for putting up with the many nights I spent working and ignoring her. Without her understanding I would never have been able to complete this project.

TABLE OF CONTENTS

INTRODUCTION

Football game scouting is that activity which coaches go through to learn the tendencies, strengths, and weaknesses of future opponents. The most used and most effective methods are the taking of still or motion pictures and/or notes by persons attending games played by the opponent against other teams. The pictures and notes are then analyzed and used to make game plans to best compete with the teams scouted.

This thesis project is designed to improve the scouting procedure by setting part of it up in computer form. It is more flexible, comprehensive, and rapidly available than the prevoius manual system, and thus it is a great time saver in game plan preparation.

My thesis consists of an automated football scouting system called SCOUT primarily designed for the University of Montana coaches and team. However it may be applied to any professional, college, or even high school football program with only minor adjustments. The backbone of the system is dependent on a thorough scouting activity of the opponents offensive and defensive plays, along with a comprehensive study of the home team's own offensive and defensive moves.

The function of this program is to take the data, collected by the coach, and analyse it and return it in a form that will allow the coaches to easily construct a game plan for playing against the scouted team.

The University of Montana coaching staff indicated a great interest in this project and were anxious to contribute to its development for use as soon as possible. The program is now in working order and is currently being used by the University coaching staff.

Football coaches, players, and fans over the years have created a jargon for the game. These language expressions belong to the game of football, and therefore will be used in this project and thesis where they seem to work best. A glossary of these terms can be found at the end of the thesis.

JUSTIFICATION

The use of computers for the purpose of collecting and analyzing data has changed the scouting methods of most professional and college athletic teams. The collecting and analyzing of data that took days when done by hand can be done in a matter of a few hours when the process is automated. In addition to speeding up the scouting process, automation also allows much more data to be collected and analyzed. These data are not only for other teams but also for the users team, and for future players that the coaching staff may seek to recruit.

The University of Montana had no automated scouting system to collect and analyze data about future opponents or the Grizzlies. They were collecting the data and analyzing it but it was all being done by hand and it took approximately three man days to collect and analyze. I felt that this was an unnecessary burden on the coaching staff and consumed valuable time that could better be used working directly with the players.

With this system the coaches can enter the data from films into the computer and the computer will do all the analyzing of the data automatically. The charts containing the information that the coaches need will then be rapidly available for use in coaching and game plan preparation.

An automated system will provide the coaches with more information in about one-fourth of the time (see fig.1). It will also make it much easier to add new information to a scouting report that has already been done at some earlier time.

Activity	Time		
	by hand	using SCOUT no terminal	using SCOUT with terminal
A. Transferring play information from film to paper	10 hr	10 hr	----
B. Travel to and from terminal room	----	.5 hr	----
C. Transferring play information from film to computer	----	----	6 hr
D. Transferring play information from paper to computer	----	6 hr	----
E. Sort and analyze information on paper	14 hr	----	----
F. Put necessary charts on paper	6 hr	----	----
G. Request necessary charts from computer	----	.5 hr	.5 hr
H. Pick up charts from computer center	----	.5 hr	.5 hr
TOTALS	30 hr*	17.5 hr**	7 hr**

* These figures are estimates supplied to me by the University of Montana coaching staff.

** These figures were calculated by entering and analyzing 95 plays (approximately one game).

FIG.1

ANALYSIS

INTRODUCTION

The analysis phase of building a computer system to automate a manual process is finding out exactly how a job is being done without the computer and then deciding what steps a computer must take to accomplish the same task. There are several methods of doing this, including the one I shall describe in the following section of this paper.

My analysis included several visits with the University of Montana coaching staff, writing to professional football teams, using my own knowledge of the game of football, and using both physical and logical data flow diagrams.

I set up an appointment to meet with the University of Montana coaching staff to work out the details about exactly what they wanted the capabilities of such a program to be. At this meeting I found that they knew very little about what could be done with an automated system.

I also decided to contact professional football teams to ascertain what their scouting programs are like. Only five of the twenty-eight teams that I contacted answered, but I was able to collect some very valuable information from their replies. This information included what the most important comparisons are and how much value should be placed on each comparison in respect to the overall scouting report.

I set up another appointment with the University of Montana coaching staff and asked them to show me exactly what data they had been collecting and how they had been analyzing the data by hand. From information gathered at this meeting I was able to get a rough idea of what they wanted. When I informed them that I felt I could write a program that would provide them with much more information in less time, they seemed to like the idea and told me to do it the way I felt it should be done.

With this information I did some preliminary analysis of the project and made some preliminary physical data flow diagrams. Using the physical data flow diagrams in addition to my personal knowledge of football scouting I made some preliminary logical data flow diagrams. Examples of data flow diagrams can be found in appendix I. I took these diagrams to the coaching staff to see if they felt this was the way they would like the program to work.

Using information received from the professional teams, I rewrote the data flow diagrams to include some of the things they were doing that could make the Grizzly system more useful. These new data flow diagrams led to a set of specifications and requirements.

While writing the code for the scouting program it became so large as one program that it had to be divided into two programs. I called these programs d-SCOUT (the defensive program) and o-SCOUT (the offensive program). By this time I had enough of an idea about the overall program that I did not need new data flow diagrams to complete it.

Because of the high school recruiting duties of the coaching staff it began to get very difficult to find any of them in town or with time to give me for collecting more information, or determining additional requirements. I wrote a requirements document from what I had, and finally found a coach to look at it. He told me that it was just fine with a couple of revisions he thought would be needed. With this information I was able to revise my data flow diagrams and requirements document to include the new information.

The game of football is played in a way that naturally separates the information needed for offense from that needed for defense. It is therefore necessary to divide the data items into two categories, one for offense and one for defense.

INPUT

With the information received from the coaches I had a good idea of the data items that the user would need to supply to the system to produce the charts required. Following is a list of these data items and a brief explanation of why each is necessary.

OFFENSE

1. The opponent of the team being scouted. This is necessary because the plays called are often very dependent on what the opponent does and the tendencies may not hold if our team is found to be very different from the opponent on the scouting report.

2. The hash mark from which the play originated. This is important as many teams have a tendency to do certain things from one side of the field that they never or seldom do from the other.

3. The yardage zone on the field that the play originated from. This is necessary because most teams use a completely different set of plays and philosophy while on one area of the field than when on another.

4. The down and distance of the play. This is necessary to allow the coaches to determine what the team tendencies are for the different yardage situations.

5. The formation used. This is important so that the coaches can get an idea of what plays the team likes to use from the various formations.

6. Was the play a run or a pass. This will tell the coaches in which situations the team likes to run and in which ones they prefer to pass.

7. The actual play that was run. This is the crux of the whole scouting idea. It allows the coaches to get some idea of which plays the opponents are likely to use in the various situations provided by the other data items.

8. The position of the ball carrier. This will let the user determine if the team likes to use a certain position as the ball carrier in specific situations or has a position that is used to carry the ball a large percent of the time.

9. The number of the ball carrier. This will allow the user to determine which players are used in which situations and if certain ball carriers are used more often then others in certain positions.

10. The hole to which the play is run. This will let the user know if the team has tendencies to run to certain hole in certain situations. Thereby giving the user an idea of what defenses to use.

11. Is the play to the strong or the weak side of the formation. This informs the user as to what defenses may be best to use with each formation.

12. The yards gained or lost by the play. This lets the user see which plays have been most successful and give him an idea of what the opponent is likely to want to run again.

13. The action of the quarterback on pass plays. This will help the coaches decide which defenses and stunts may work best in rushing the quarterback and covering the receivers.

14. The result of a pass play. This will tell the user which pass plays have been most successful and therefore which ones the team is most likely to use again.

15. The position from where receiver started. This will tell the coaches which receiver is most likely to have the ball thrown to him in each situation.

16. The number of the receiver. This will let the coaches know which receivers run certain patterns best and the one the quarterback likes to throw to in this situation.

17. The pattern run by the receiver. This, combined with the previous two items lets the user know what to expect from each receiver.

18. The depth of the pass thrown. This lets the user know the best coverages to use in each situation.

19. The zone on the field to which the pass is thrown. This lets the user know what coverage will probably work best in each situation.

20. The drive number in which the play occurred. This lets the user know at what point in the game the given play took place.

21. The play number in the drive. This allows the user to see which plays are used to set up other plays.

DEFENSE

1. The opposing team. As with offense this is necessary because what a team does is often dependent on what the team they are playing does.

2. The hash mark from where play started. This lets the user know if the team has different tendencies on different sides of the field.

3. The yardage zone in which the play started. This lets the user know what the team is likely to do on defense at different locations on the field.

4. The down and distance of the play. This will tell the user what defenses the team is likely to use for different down and distance situations.

5. The formation that the offense used. This will tell the coaches what defenses to look for if they use certain formations.

6. Any Motion that the offensive team used. This will tell the coaches which coverages will be use if they put men in motion.

7. The defensive front that was used. This will let the coaches know what blocking schemes they should use to best block the opposing lineman.

8. Any variations to standard fronts that are used. This will let the user know which blocking assignments need to be adjusted from those used for standard fronts.

9. The secondary coverage used. This will give the coaches an idea of which pass patterns will work best.

10. The line stunts that are used. This gives the user an idea of what sort of blocking assignment will need to be adjusted to block the stunting linemen.

11. Any blitzes that are used. This will give the coaches an idea of how many men they can put into the pass patterns and how to best plan to block any blitzing linebackers.

The coaches and I decided that the above items of information if properly analysed would provide the necessary information to create a complete scouting report.

OUTPUT

The next step in my analysis was to determine what form the output should take to be of maximum value to the coaches using the program. I talked to several of the coaches and it soon became apparent that the system must combine any or all of the data elements, and that the user should be able to select which of these he wanted each time.

It also became apparent that the system should be able to produce at least two different types of charts. One chart that would show the details of each play on the chart and another that would show at a glance the frequency with which certain items occurred in given situations.

My next step was to look at BLITZ and FOOTBALL SCOUTER two commercially available football scouting programs to get some ideas about how the output charts should be formatted. I showed these formatting ideas to the coaching staff and received their approval on two of the chart formats.

Examples of the two types of output and an explanation of each follows.

```
        COVER BY HASH
     | 1 | 2 | 3 | 5 | 6 | M |MF |M2D|WMM|WM2|WM3|COM|ROB|other
--------------------------------------------------------------
  R |  1|  4|  0|  5|  3|  0|  0|  6|  5|  7| 11|  0|  3|  2 |
--------------------------------------------------------------
  M |  2|  3|  1|  4|  2|  1|  0|  5|  3|  4|  5|  2|  0|  0 |
--------------------------------------------------------------
  L |  0|  1|  0|  6|  3|  0|  0|  4|  4|  5| 10|  0|  1|  1 |
--------------------------------------------------------------
othr|  0|  0|  0|  0|  1|  0|  0|  0|  0|  0|  2|  0|  0|  1 |
--------------------------------------------------------------
      3   8   1  15   9   1   0  15  12  16  28   2   4   4
     2%  7%  1% 13%  8%  1%  0% 13% 10% 13% 23%  2%  3%  3%
THERE ARE 118 PLAYS ON THIS CHART
```

FIG.2

Figure 2 is an example of a MATRIX type chart. The top line "COVER BY HASH" is the name of the chart and tells which two items are being compared. The first part of the name ("COVER" in the above example) is the items in each column and the second ("HASH" in the above example) is the items in each row. COVER in this heading means the secondary coverage used by the team being scouted and HASH is for the hash mark where the play originated. The actual codes such as WWM and ROB are the ones I obtained from the Grizzly coaches and would probably be different for another coaching staff.

Line two is the actual name of each column, in this case the name of each of the coverages used. These names are the ones that the University of Montana uses to describe the different defensive coverages.

The first item in the next four rows are the names of the items in each row, in this case the hash mark. L standing for the left hash mark, R standing for the right hash mark,and M standing for the middle of the field. The numbers that follow are the number of times that the items in the intersecting row and column occurred in the same play.

The last column and row are to indicate any occurrences of items that are not included in the standard items.

The numbers in the seventh row of this example indicate the total number of times that the item in each column occurred. The next row is the percentage of times that each item occurred.

The final line gives the total number of plays on the chart.

The actual size of each chart will vary according to the number of items of each type to be compared, but the general format for all matrix charts is to be the same.

It was decided that the following comparisons should be available on the matrix charts.

OFFENSE

1. The number of times that each hole is run for each down and distance.

2. The number of times that each hole is run from each hash mark.

3. The number of times each hole is run from each formation.

4. The number of times each play is used from each down and distance.

5. The number of times each pass action is used from each hash mark.

6. The number of times each pass route was used from each hash mark.

7. The number of times each formation was used from each hash mark.

8. The number of times each pass action was used from each yardage zone.

9. The number of times each pass route was used from each yardage zone.

10. The number of times each pass action is used from each down and distance.

11. The number of times each pass route was used from each down and distance.

12. The number of times each formation was used from each down and distance.

13. The number of times that each play was run from each formation.

DEFENSE

1. The number of times each secondary coverage is used for each front.

2. The number of times each secondary coverage is used for each offensive formation.

3. The number of times each secondary coverage is used for each yardage zone.

4. The number of times each secondary coverage is used for each hash mark.

5. The number of times each secondary coverage is used for each down and distance.

6. The number of times each front is used for each down and distance.

7. The number of times each front is used for each offensive formation.

8. The number of times each front is used for each yardage zone.

9. The number of times each front is used for each hash mark.

For the list charts it was decided to let the user select each chart on any number of items he might desire. He will be asked for the number of items he wishes to key and then he will be asked for the key value for each item. In this way any conceivable game situation that is wanted can be created for checking. Any of the items that were entered about the plays can be keyed and the user will then select a particular value for the chosen item.

```
     CHART KEYED ON HASH MARK
   |  |  | |    | d|r|    |  |  | | |   | |r |  |  | |   |
   |dn| n| |    | i|u|    |p |n | | |   |r|en| p|  |p|  r|
   |ru|pu|h|    |ds|n|    |o |u |h|s|   |e|cu|pa|pd|!|p e|gl
   |im|lm|a|    |ot|!|    |s |mb|o| |p- |c|em|at|ae|z|a s|as
   |vb|ab|s|    |wa|p|    |! |bc|l|!|   |!|ib|st|sp|o|s u|is
opp|ee|ye|h|form|nn|a|play|b |e |e| |act|p|ve|se|st|n|s l|ns
   | r| r| |    | c|s|    |c |r | |w|   |o|er| r| h|e|  t|
   |  |  | |    |&e|s|    |  |  | | |   |s|r | n|  | |   |
------------------------------------------------------------
MSU| 1| 2|R|PRI |2M|R|34  |B |22|4|S|   | | |  |  | |   | 6
------------------------------------------------------------
MSU| 2| 1|R|TWLI|1L|P|107X|  |  | |W|700|Y|89|1 |M |3|COM|12
------------------------------------------------------------
MSU| 3| 3|R|SPLB|2L|P|941 |  |  | |S|900|Z|88|1 |VL|8|INC| 0
------------------------------------------------------------
 AVERAGE = 6.00       MODE = 0   TOTAL NUMBER OF YARDS = 18
 MEDIAN  = 6          MEAN = 6.00
 NUMBER OF KEYED PLAYS = 3
 THIS IS 5% OF ALL THE PLAYS
 TOTAL PLAYS ENTERED = 55
 STANDARD DEVIATION IS 3.7
```

FIG.3

Figure 3 is an example of a list chart. Basically it is a list of all the plays that have been entered that meet the requirements that the user has requested.

The first row tells the name of the chart and lets the user know on what item the chart was keyed. If more than one key is used the last key will be the one for which the chart is named. For example if the user asked for three keys such as: drive number, play number, and hash mark the chart will be named "CHART KEYED ON HASH MARK".

The next eight rows are the names of the items that appear in each column. Because of the width of some of the colunms some of the names are written vertically.

Each of the following rows contains the information about an individual play that meet the requirements for this chart. In this example the chart was keyed on the right hash mark and therefore all the hash column entries are "R".

The next rows contain the statistical information about the plays in the list. These statistics are figured on the gain or loss value of the plays.

1. Average: This is the average gain or loss for the plays on this list.
2. Mode: This is the gain or loss value that appears most frequently in the list.
3. Total number of yards: This is the total number of yards gained or lost by all the plays in the list.
4. Median: This is the middle play in the list if the plays are taken in gain loss order from lowest to highest.
5. Mean: This is the average gain or loss if the lowest and the highest values are not considered.

6. Number of keyed plays: This is the number of plays on this chart.
7. Percent of all plays: This tells what percentage of all the plays entered are included in this list.
8. Total plays entered: This tells the total number of plays that have been entered about the team being scouted.
9. Standard deviation: This is the standard deviation of the gain or loss values for the plays in this list.

The length of this chart will of course vary depending on how many plays meet the necessary keyed values. In the event that there are no plays that contain all the keyed values this will be reported to the user.

The user is to be given two options for the order of the plays in the list charts. He will be given the choice of having them in the order they were entered or in ascending order according to the gain of the play.

USER INTERFACE

At the start of the program the user is to be asked for the name of the team being scouted. This name will then be used to create the names for the input files to be read and the output file to be produced.

Next the user will be prompted for each item to be entered about any new plays he wants to put into the scouting report. The program is designed to allow the user to enter these item while viewing the game films. This is to eliminate the time that it would take to write them on parer and then enter then into the program. These new plays will be added to any old plays that were read from a file.

The user is to be given a choice as to whether he wants matrix charts or list charts. After selecting one, and having the charts produced he, will again be given this choice in case he also wants the other type.

A menu of the possible combinations for matrix charts is to be given to the user and he can then choose which ones he wants by number. The user will also be given a menu of items that can be used as keys for the list charts he wishes to have produced. For the particulars of this menu please refer to appendix VII and VIII.

I used the information that I had gathered and wrote a final requirements document for the system. I presented this requirements document to the coaching staff and they gave their approval.

DATA FLOW DIAGRAMS AND DATA DICTIONARY

With the requirements clearly defined I was able to draw my logical data flow diagrams and make a data dictionary. The data flow diagrams may be found in appendix I and the data dictionary in appendix II.

Data flow diagrams are a pictorial representation of the flow of data through a program from the initial input to the final output. They deal strictly with the flow, and transformation of data and control is not considered.

Data flow diagrams consist of the following elements:

1. Circles which represent processes where data may be transformed from one form to another. The name in the circle indicates what the process does.

2. Vectors which represent data flow from one process to another. The name of the vector is the data being transported.
3. Straight line segments which represent file from which may be read or to which it may be sent.
4. Rectangular boxes which represent data sourses such as the user.

Data flow diagrams are drawn in such a way that the first or top level ones show the overall system as one process. The succeeding ones divide the process into smaller processes until the lowest level diagrams show a process representing a single transformation of data. The number of each chart is the number of the process that is broken down in the diagram. The number of incoming and outgoing data flows on a chart must always be the same as the number coming and going from the process it represents.

An explanation of a representative diagram such as (DIAGRAM 1 BUILD- DEFENSIVE-CHARTS, page 36) follows.

This diagram is the first break down of the whole defensive scouting system shown as process 1 on diagram 0. It has two inputs - Defensive-Play and Defensive-Play-Wanted here called by the alias D-P-W, and two outputs - Defensive-Matrix-Charts (D-M-C) and Defensive-List-Charts (D-L-C).

In this diagram process 1 is broken down to the three main processes that take place in the system:

process 1.1 (READ-AND-LIST-THE-DEFENSIVE-PLAYS),

process 1.2 (BUILD-DEFENSIVE-MATRIX-CHARTS), and

process 1.3 (BUILD-DEFENSIVE-LIST-CHARTS).

Each of these processes will be further broken down in lower level charts.

The chart shows that the defensive play is read into process 1.1 and it transforms this data into a defensive-play-record-list (D-P-R-L) which it then passes on to processes 1.2, 1.3, and the defensive play file.

Process 1.2 receives the defensive-play-record-list and transforms it into the defensive-matrix-chart. Process 1.3 receives the defensive-play-record-list and the defensive-play wanted and transforms them into the defensive list chart.

The data dictionary is a document that must accompany a set of data flow diagrams. It contains a precise definition of each data flow on the data flow diagrams explaining exactly what information each holds. It also contains an explanation of what transformations of data take place in each process.

With both the requirements and the data flow diagrams completed I was able to proceed with the next phase of developing the scouting system. As requirements changed and/or I ran into difficulties with the design I had to return to this phase and rework part of the analysis. The data flow diagrams and data dictionary in the appendixes are the final ones reflecting how SCOUT was finally constructed.

DESIGN

INTRODUCTION

The design phase of software development and how it applied to SCOUT will be discussed in this section.

Design is the phase of software development in which the software engineer takes the analysis of what is to be done and decides how to get the computer to do it. There are many different methods of design and no one is really best. The method I chose, which I will discuss in this section, is structured design. I chose structured design because it is a method designed to convert data flow diagrams into a satisfactory design document called a structure chart. The structure charts for SCOUT are in appendixes III and V.

During the design phase the designer must also decide on what types of data structures will be used in the program. This was done at the same time that the structure charts are drawn to assure that the modules on the structure charts can accomplish the task each has to do.

DATA STRUCTURES

The two main data structures that I decided to use in SCOUT were a linked list of records and two dimensional arrays indexed by the types of the items for which they represent values for.

Each record corresponds to one play that has been entered. The records are to consist of a field for each item that is entered about a play and one field to hold a pointer to the next record. The items that are to be entered about each play are the ones that are enumerated in the input section of analysis. The same type record can also be used to hold the values of each field of the play that the user wants in the list charts.

To generate the list charts, a "template" for the type of play wanted and assigning a not-specified flag to each of the fields that the user has not keyed can be made using one of the records. This will allow the system to make a single pass through the list of plays and print those plays that match the template.

The two dimensional arrays are to be used to hold the values for the matrix charts. By using arrays that are indexed by the possible values in the necessary fields for these charts the values can be assigned to all of them with a single pass through the list of plays. Then each array can be used to supply the values to the corresponding chart.

STRUCTURE CHARTS

Having the main data structures for the system I drew the necessary structure charts to define the modules of the system. These structure charts were drawn using transformational design and analysis.

A structure chart is a pictorial diagram of the system showing each of the modules, the interfaces between the modules, and the hierarchy of the system. The basic elements of a structure chart are:

1. A module is represented by a rectangular box. The lowest level modules will be procedures and functions in the program. This box will contain the module name and number for referencing the design document.
2. The connection between modules represented by a vector, ---->, pointing from the calling module to the called module.
3. A diamond around the tail of a vector inside the calling module indicates that a decision is made as to whether the module should be called or not.
4. A curved arrow around the tail in the calling module indicates that the call is being made in a loop and may be made several times.
5. Data passed from one module to another, represented by an arrow with an empty circle for a tail. O----->.
6. Control information passed from one module to another, represented by an arrow with a filled circle for a tail, O---->.
7. A short description of what the data or control is written beside the arrow.

To create the first level structure charts a designer analyses the data flow diagrams and determines which ones are afferent (input), which are efferent (output), and which are transformation modules. One of the transformation modules is chosen as the driver for the system and placed at the top of the structure chart while the afferent ones are usually place below and to the left of it and the efferent ones are usually placed below and to the right.

Once the first level structure chart is completed it may become necessary to divide some or all of the modules into smaller subordinate modules. This process should continue until each module represents a single task and can be easily transformed into pseudocode to describe in some detail what each module does. Each of the lowest level modules should be equivalent to an independent function or task in the code for the system.

By applying this methodology to the data flow diagrams that were produced for SCOUT, I was able to draw the structure charts for the system. These structure charts may be found in appendix III and V. A brief explanation of each module follows. The complete algorithm for each is included in appendix III and V. The number in front of each module refers to the module number on the structure chart.

d-SCOUT

Chart 0.

1. DRIVERS: This is a composite module that includes all the modules on chart 1.

2. REED-D-PLAYS: This module reads the plays to be added to the scouting report from the terminal.

3. REED-OLDD-PLAYS: This module reads any plays that are in a file and are to be included in this scouting report.

4. DLIST-CHART: This module finds out which plays the user include in any list charts that are produced.

5. PRINT-LIST-CHART: This module writes the list of plays that match the values for which the user asked.

6. D-HEADING: This module writes the heading on any list charts that are produced.

7 - 15. These are the modules that write the matrix charts and include all the modules on chart 7-15.

16. BUILD-D-LIST: This is the module that puts the plays read in by reed-d-plays and reed-oldd-plays into a list that can be accessed by other modules.

17. DLIST-CHART-TRAILER: This module puts the information necessary at the end of the list charts.

18. DRAW-LINE: This module is used to draw any lines needed on any of the output charts.

19. MAKE-D-FILE: This module writes a file containing all the plays that have been input to the system.

20. INIT-FIELD: This module initializes the name of the charts to spaces before each new chart is requested.

Chart 1.

1.1 MAIN: This is the module that introduces the system and gets the general information about the team being scouted.

1.2 INIT-FILENAME: This module initializes the input and output file names to spaces.

1.3 FILENAME: This module puts the proper name in the input and output filename variables.

1.4 SED-VARS-AND MATRIX-VALUES: This module accesses the list of plays and set the variables to the variables to the proper values according to the information in the list. It consists of the modules on chart 1.4.

1.5 WHAT-DCHART: This module finds out what type of charts the user wants and calls the modules to produce the required charts.

1.6 MATRIX-DCHART: This module finds which matrix charts are wanted and calls the modules to write them.

Chart 1.4

1.4.1 SET-MATRIX-ARRAY-VARS: This module looks at each play list and increments the elements of the arrays that are indexed by the values of the play.

1.4.2 INIT-MATRIX-ARRAYS: This module initializes the elements of the matrix arrays to 0.

1.4.3 ASSIGN-NAMES: This module put the proper names in the name arrays.

1.4.4 COLUMN-TOTALS: This module finds the total number of times that the event in each column of the matrix arrays occurs.

1.4.5 COLUMN-PERCENTS: This module finds the percentage of times that each column event occurs.

Chart 7-15

7.1 PRINT-FRONTBYHASH-AR: This module has the front by hash array printed.

7.2 PRINT-COVER-HEADING: This module puts the heading on the cover charts.

7.3 PRINT-FRONT-HEADING: This module puts the heading on the front charts.

7.4 PRINT-FRONT-TOTALS: This module prints the column totals for the front charts.

7.5 PRINT-COVER-TOTALS: This module prints the column totals for the cover charts.

7.6 PRINT-FRONT-COLUMN-PERCENTS: This module prints the percentages for the columns on the front charts.

7.6 PRINT-COVER-COLUMN-PERCENTS: This module prints the percentages for the columns on the cover charts.

NOTE: Module 7.1 is a representative module of modules 7 through 15 on chart 0. Modules 7 through 10 which print front charts call modules 7.3 and 7.4. Modules 11 through 15 which are for cover charts call modules 7.2 and 7.5.

o-SCOUT

Chart 0.

1. DRIVERS: These are the main driver modules for the and are expanded on chart 1.

2. REED-O-PLAYS: This module reads the plays from the terminal.

3. REED-OLDO-PLAYS: This module reads the plays from a file.

4. OLIST-CHART: This module finds which plays the user wants included in the list charts.

5. PRINT-LIST-CHART: This module prints the list charts the user wants.

6. O-TRAILER: This module puts the trailer on the end of the list chart.

7 - 19. These are the modules that print the matrix charts. The are expanded on chart 7-19.

20. INIT-FIELD: This module initializes the array that holds the name of the list charts.

21. MAKE-O-FILE: This module writes the output file of plays.

22. BUILD-O-LIST: This module makes the list of plays in the order they are entered to be accessed by other modules.

23. SORTED-LIST: This module makes the list of plays in gain loss order.

24. O-HEADING: This module writes the heading on the list charts.

25. DRAW-LINE: This module writes lines on the output charts.

26. INIT-GAIN: This module initializes the elements of the gain loss array to 0.

27. INIT-FILENAME: This module initializes the variables that hold the file names to spaces.

28. FILENAME: This module puts the names of the input and output files in the proper variables.

Chart 1

1.1 WHAT-OCHART: This module finds out what kind of output charts the user wants.

1.2 SET-VARS-AND-MATRIX-VALUES: These modules are expanded on chart 1.2.

1.3 MATRIX-CHARTS: This module finds out which matrix charts the user wants.

Chart 1.2

1.2.1 SET-MATRIX-ARRAY-VARS: This module scans the list of plays and puts the proper values in the matrix variables and arrays.

1.2.2 INIT-MATRIX-ARRAYS: This module initializes the elements of the matrix arrays to 0.

1.2.3 ASSIGN-NAMES: This module assigns the names to the name arrays.

1.2.4 COLUMN-TOTALS: These modules find the totals for the elements in each column of the matrix charts.

1.2.5 COLUMN-PERCENTS: These modules find the percentage of times the elements of each column are used.

1.2.6 PLAY-TOTALS: This module finds the total number of times that each play is used.

1.2.7 PLAY-PERCENTS: This module finds the percentage of times each play is used.

Chart 6.

6.1 O-MATRIX-TRAILER: This module calls the modules that print the trailer on the list charts.

6.2 CALCULATE-MODE: This module finds the mode for the gains of all the plays on the chart.

6.3 CALCULATE-MEAN: This module finds the mean for the gains of all the plays on the chart.

6.4 CALCULATE-STANDARD-DEVIATION: This module finds the mode for the gain of all the plays on the chart.

6.5 CALCULATE-MEDIAN: This module finds the median for the gain of all the plays on the chart.

6.6 CALCULATE-AVERAGE: This module finds the average for the gains of all the plays on the chart.

Chart 7-19

7.1 MATRIX-CHARTS: These are the modules that have the matrix charts the the user wants printed.

7.2 PRINT-MATRIX-HEADER: These modules print the headers on the matrix charts.

7.3 PRINT-COLUMN-TOTALS: These modules print the column totals on the matrix charts.

7.4 PRINT-COLUMN-PERCENTS: These modules print the column percents for the matrix charts.

Note: There are 13 of these modules, one for each type matrix chart that is available.

IMPLEMENTATION

INTRODUCTION

The implementation of a computer system consists of the coding (actual writing of the computer code), testing, and usage of the program. I wrote SCOUT in Pascal so that it could be used by the University of Montana football team, and installed on the university's DEC 2060 computer. Moreover, I wrote the program in such a manner that it would be portable enough to be installed on any of several micro computers which might be used in high school coaching.

CODING

The writing of the Pascal code for SCOUT posed very few problems. The algorithms that had been written during the design phase of the project made coding a relitivly easy task. One problem that did occur however in trying to make it usable by as many football coaches as possible, was that most football teams use different terminology for naming plays, formations, front, coverages, etc. I used constants to define these things to allow for easy changing of terms to make the program usable by more than just the University of Montana team.

The other main coding problem arose from the fact that I originally designed the program with a different set of data structures and spent several months coding to meet this standard. The program was actually finished and running with the original data structures before I decided to change to the ones that are now being used. The changes were made to make the program smaller to allow it to be used on micro computers.

The changes that were made concerning the data structures required several additional months of effort. The extra months were required for redesign, coding, and testing.

I chose not to use parameters to pass global variables to keep the amount of memory space required as small as possible in order to be able to install the system on micro computers. The global variables that are used or modified are documented in each procedure. <u>SCOUT</u> could now be written as one program but the university coaches like it as two so I left that way.

PORTABILITY

The other reason for leaving it as two separate programs is for portability to smaller computers. In the present state, the two programs could easily fit into any micro computer with 64K or more memory.

I estimated the memory requirements of the programs with the assumption that 50 plays would be entered for each game. I made the further assumptions that each character would require one byte of memory, each integer would require two bytes, and each boolean variable would require 2 bytes.

The two largest consumers of memory in the system are the list of play records and the matrix chart variables. The memory requirements for the system were estimated in the following manner:

1. A defensive play record contains eleven items with a total of 61 characters and one pointer. This is a total of 63 bytes. A linked list of these records is made with one record for each play for a total of 3150 bytes needed for fifty plays.
2. The program contains nine matrix chart arrays containing a total of 1695 integers. This number is constant and not dependent on the number of plays entered. Each integer requires two bytes of memory for a total of 3390 bytes required for the matrix arrays.
3. The other memory requirements were analysed in the same manner and I found that they would require approximately 1000 bytes.

Pascal seems to have no standard method for handling interactive input and output, Therefore if the system is to be implemented on a micro computer the input and output may have to be changed. Other than this the program is written in standard Pascal with no programming tricks or things that may keep it from running on any computer with a

Pascal compiler.

TESTING

The testing of SCOUT was done in three phases. First, preliminary testing was done as it was being coded. Second both my son and I tested after coding was complete. Third, it was tested by the University of Montana football coaches as they used it for the second half of the 1984 season.

During the first phase, I carefully tested each procedure or function by itself to be sure that it would perform as expected before making it a part of the program. If it passed the initial tests I then added it to that part of the program that was finished and then retested the whole program again. I would not start to code a new procedure or function until the ones coded before all worked together properly.

For phase two after the coding was complete my son and I spent two weeks entering plays and generating charts to be sure that we obtained the desired results from as wide a variety of inputs as possible. At this point we also entered many errors into the data to make sure that the error recovery would keep the previously entered data from being modified or destroyed.

For the third phase of testing I installed the program in the University of Montana's football team's computer area and the coaches used it to scout several games. They were told that the system had not been completely tested and that some errors might arise. Their instructions were to inform me if the system had any problems and I would try to correct them.

Errors were found and corrected during each phase of the testing. Each phase produced fewer errors than the previous phase and during the last three games that the University football team used SCOUT no errors were found.

As with any computer program I can not guarantee that there are no bugs left in the system. I do feel however that the program has been tested as much as can be expected and should be relatively free of bugs and quite reliable.

DIAGRAM 0
SCOUT

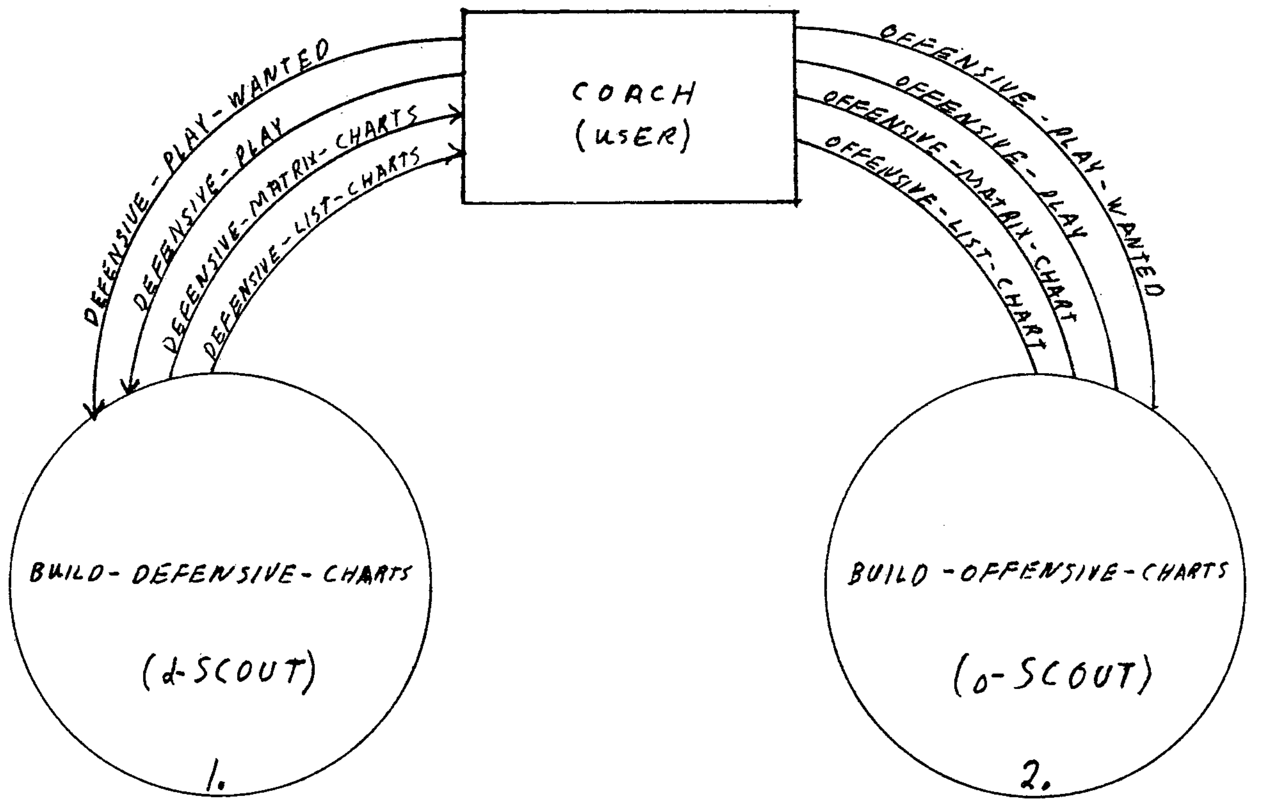


DIAGRAM 1
BUILD-DEFENSIVE-CHARTS

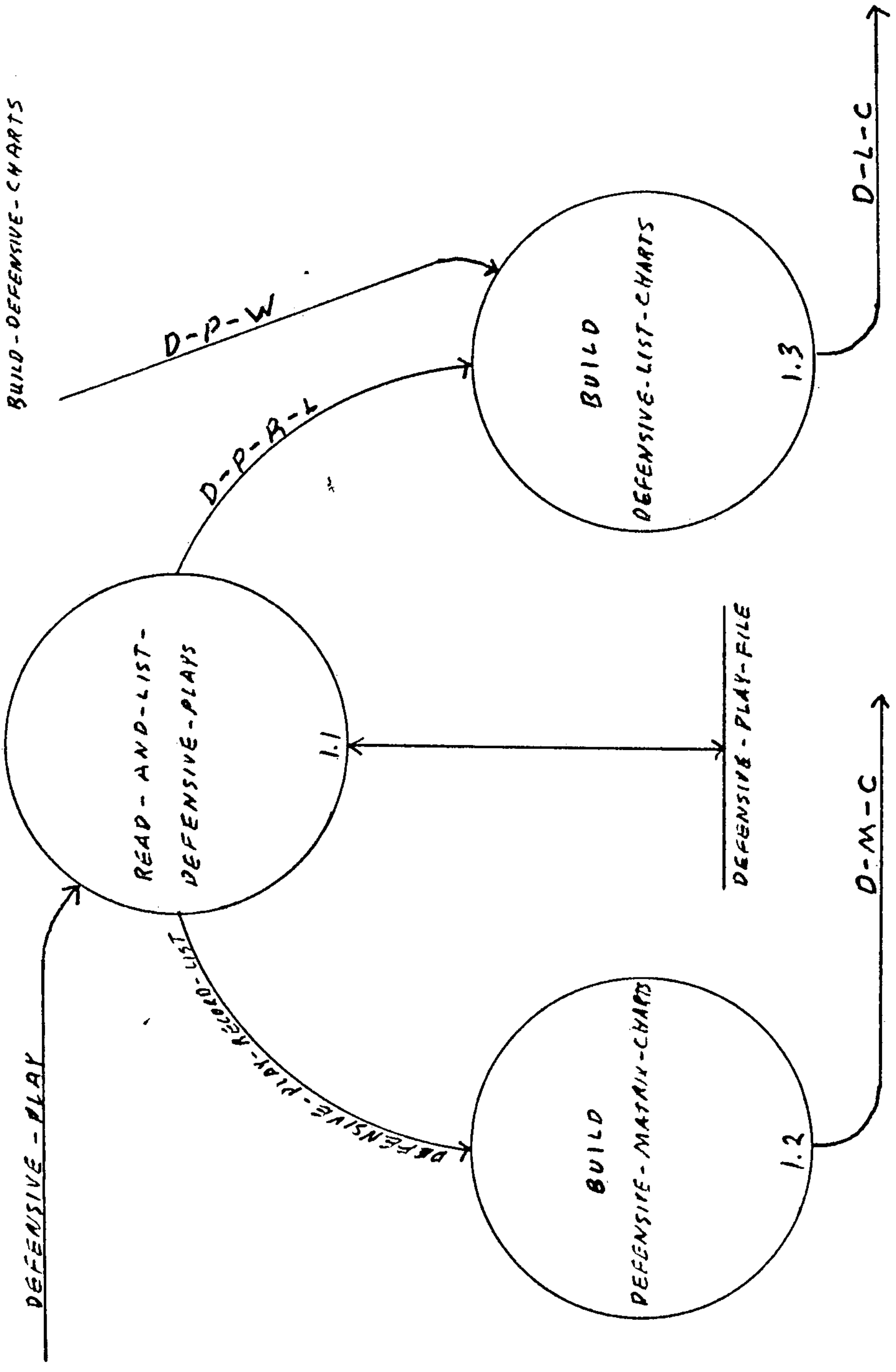

DEFENSIVE-PLAY
READ-AND-LIST-DEFENSIVE-PLAYS
1.1
D-P-W
D-P-R-L
BUILD DEFENSIVE-LIST-CHARTS
1.3
D-L-C
DEFENSIVE-PLAY-FILE
DEFENSIVE-PLAY-RECORD-LIST
BUILD DEFENSIVE-MATRIX-CHARTS
1.2
D-M-C

DIAGRAM 1.1

READ-AND-LIST-DEFENSIVE-PLAYS

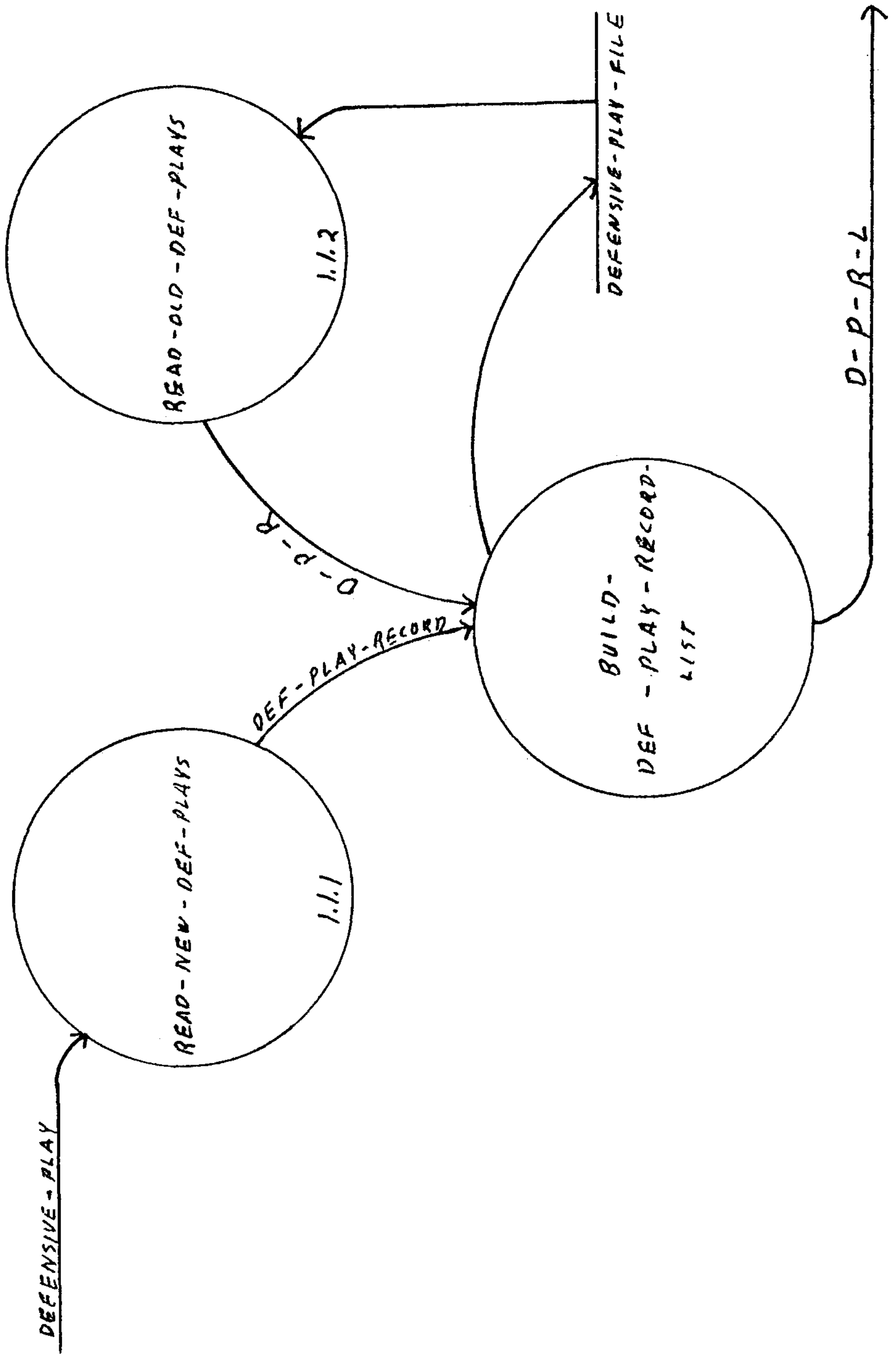


DIAGRAM 1.2
BUILD-DEFENSIVE-MATRIX-CHARTS

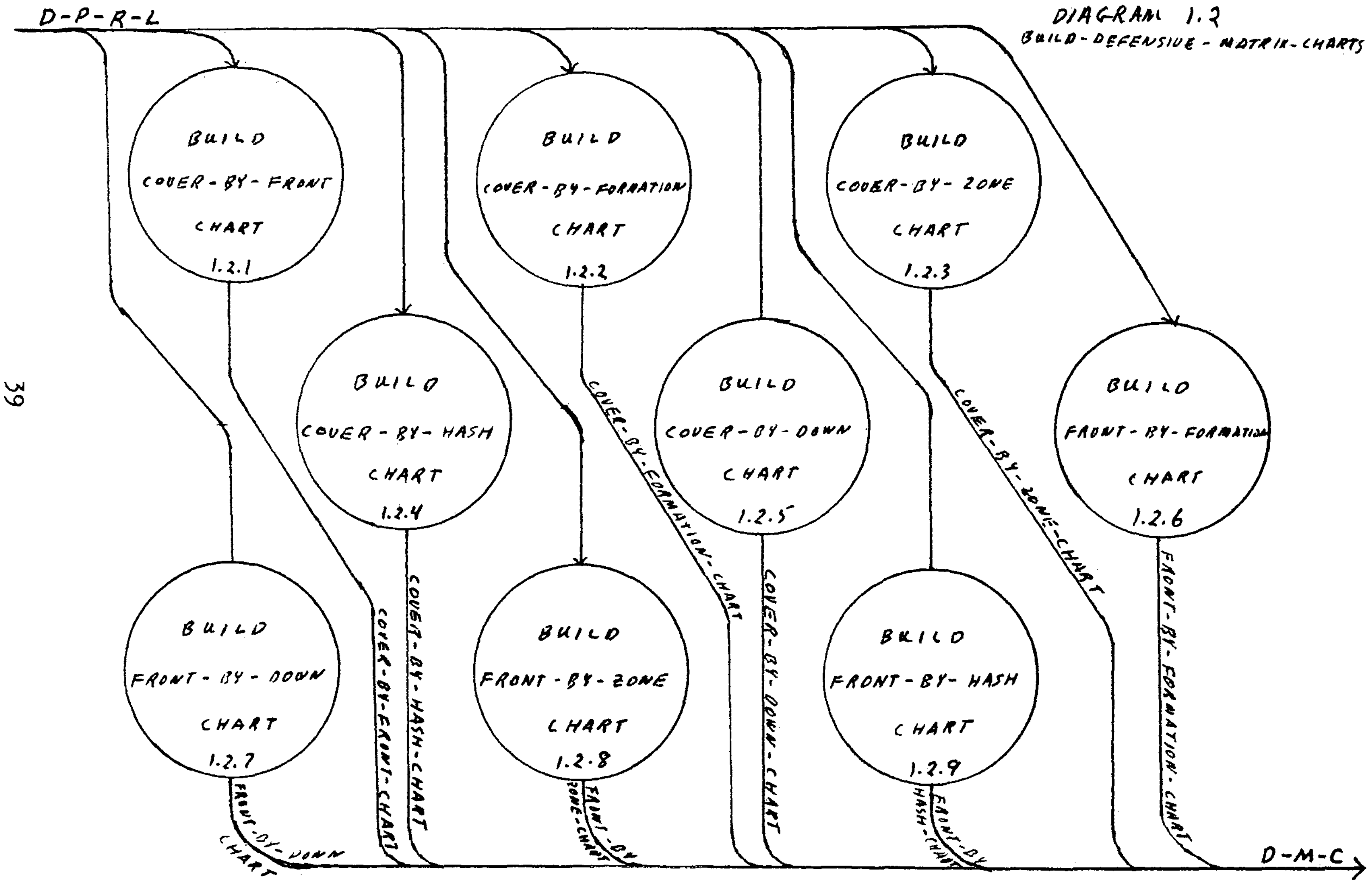
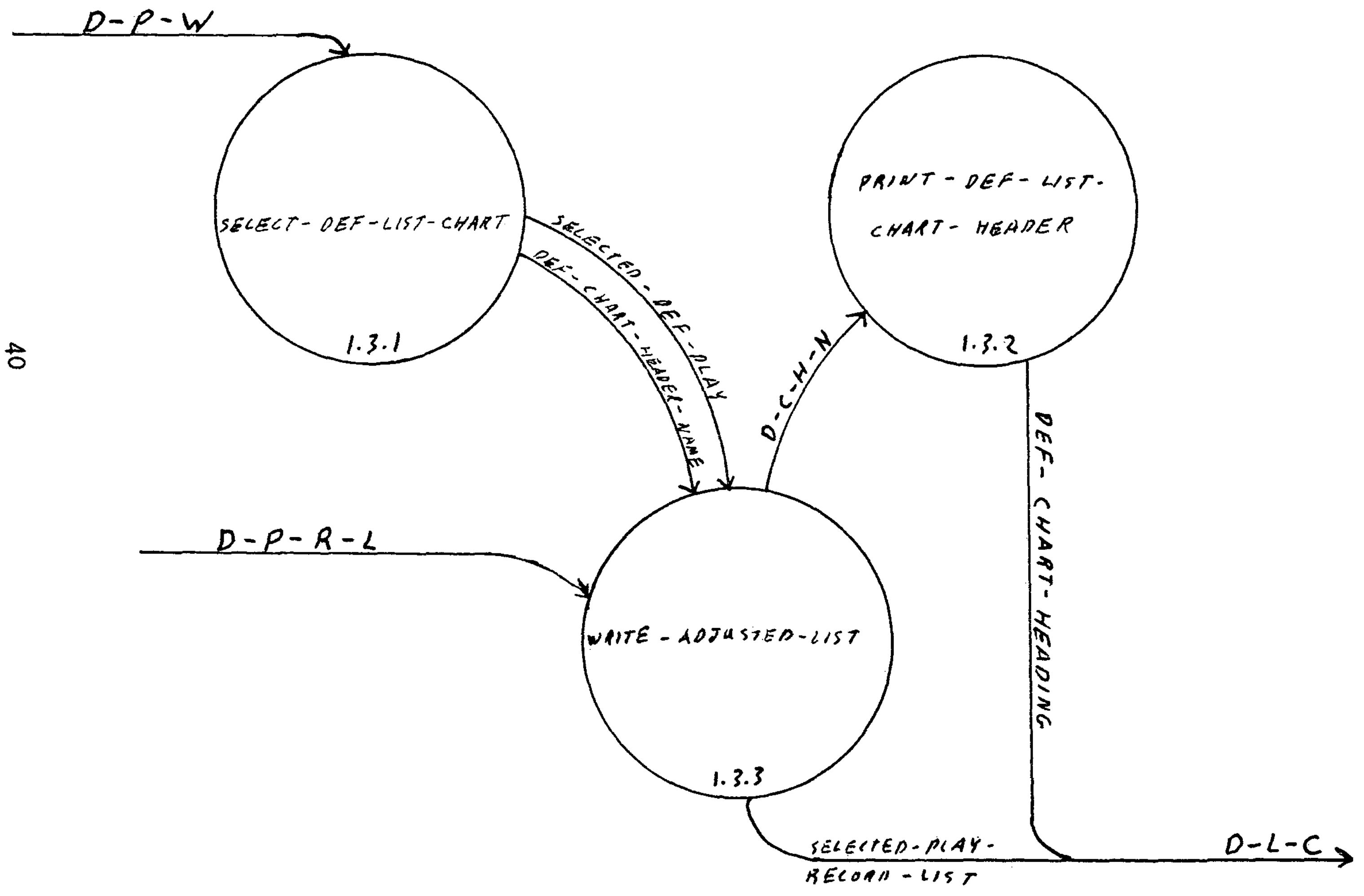

D-P-R-L
BUILD COVER-BY-FRONT CHART 1.2.1
BUILD COVER-BY-FORMATION CHART 1.2.2
BUILD COVER-BY-ZONE CHART 1.2.3
BUILD COVER-BY-HASH CHART 1.2.4
BUILD COVER-BY-DOWN CHART 1.2.5
BUILD FRONT-BY-FORMATION CHART 1.2.6
BUILD FRONT-BY-DOWN CHART 1.2.7
BUILD FRONT-BY-ZONE CHART 1.2.8
BUILD FRONT-BY-HASH CHART 1.2.9
COVER-BY-FRONT-CHART
COVER-BY-HASH-CHART
COVER-BY-FORMATION-CHART
COVER-BY-DOWN-CHART
COVER-BY-ZONE-CHART
FRONT-BY-FORMATION-CHART
FRONT-BY-DOWN CHART
FRONT-BY ZONE-CHART
FRONT-BY HASH-CHART
D-M-C

DIAGRAM 1.3
BUILD-DEFENSIVE-LIST-CHART



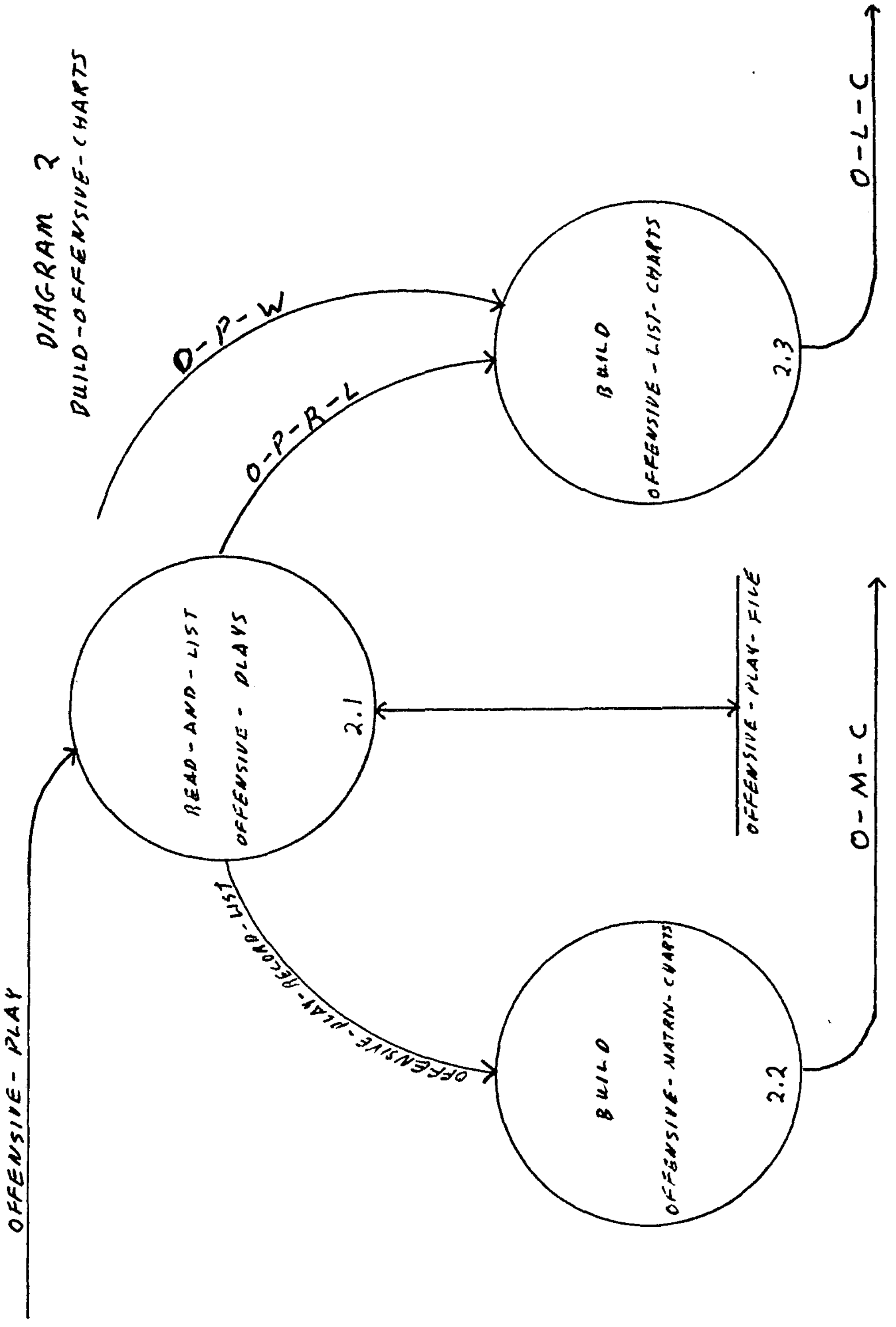


DIAGRAM 2
BUILD-OFFENSIVE-CHARTS
OFFENSIVE-PLAY
READ-AND-LIST OFFENSIVE-PLAYS
2.1
O-P-W
O-P-R-L
BUILD OFFENSIVE-LIST-CHARTS
2.3
O-L-C
OFFENSIVE-PLAY-FILE
OFFENSIVE-PLAY-RECORD-LIST
BUILD OFFENSIVE-MATRX-CHARTS
2.2
O-M-C

DIAGRAM 2.1

READ-AND-LIST-OFFENSIVE-PLAYS

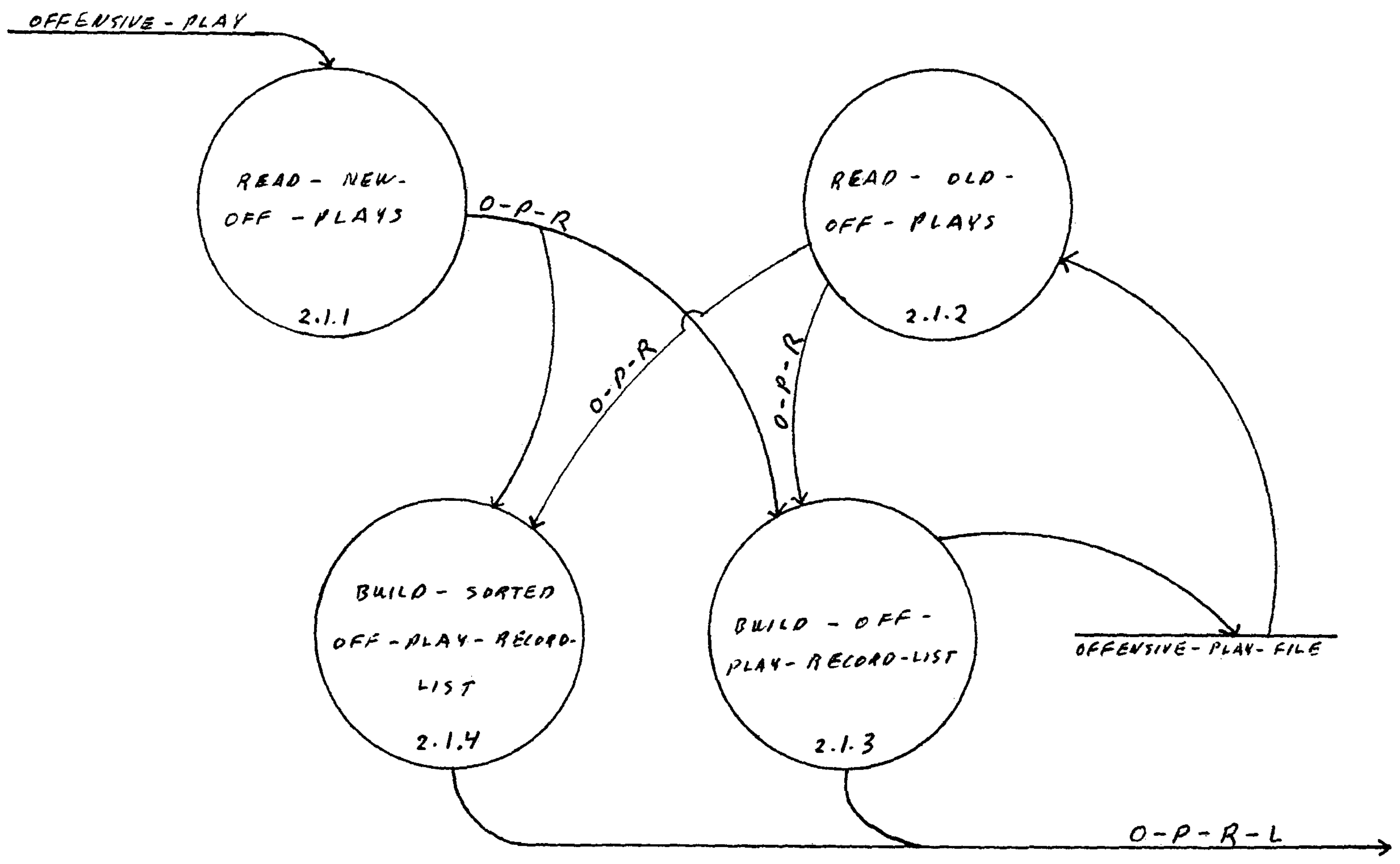


DIAGRAM 2.2(A)
BUILD-OFFENSIVE-MATRIX-CHARTS

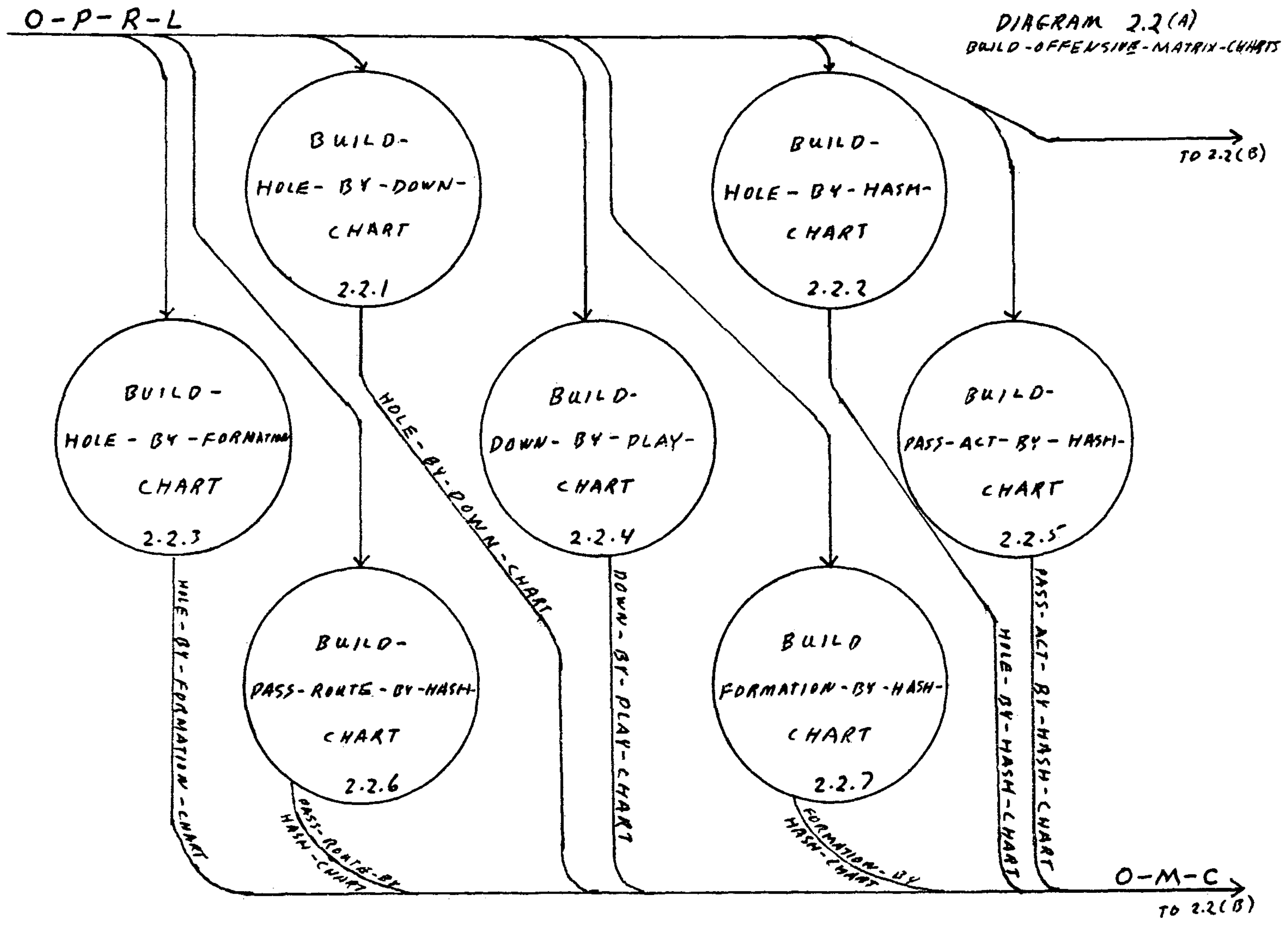
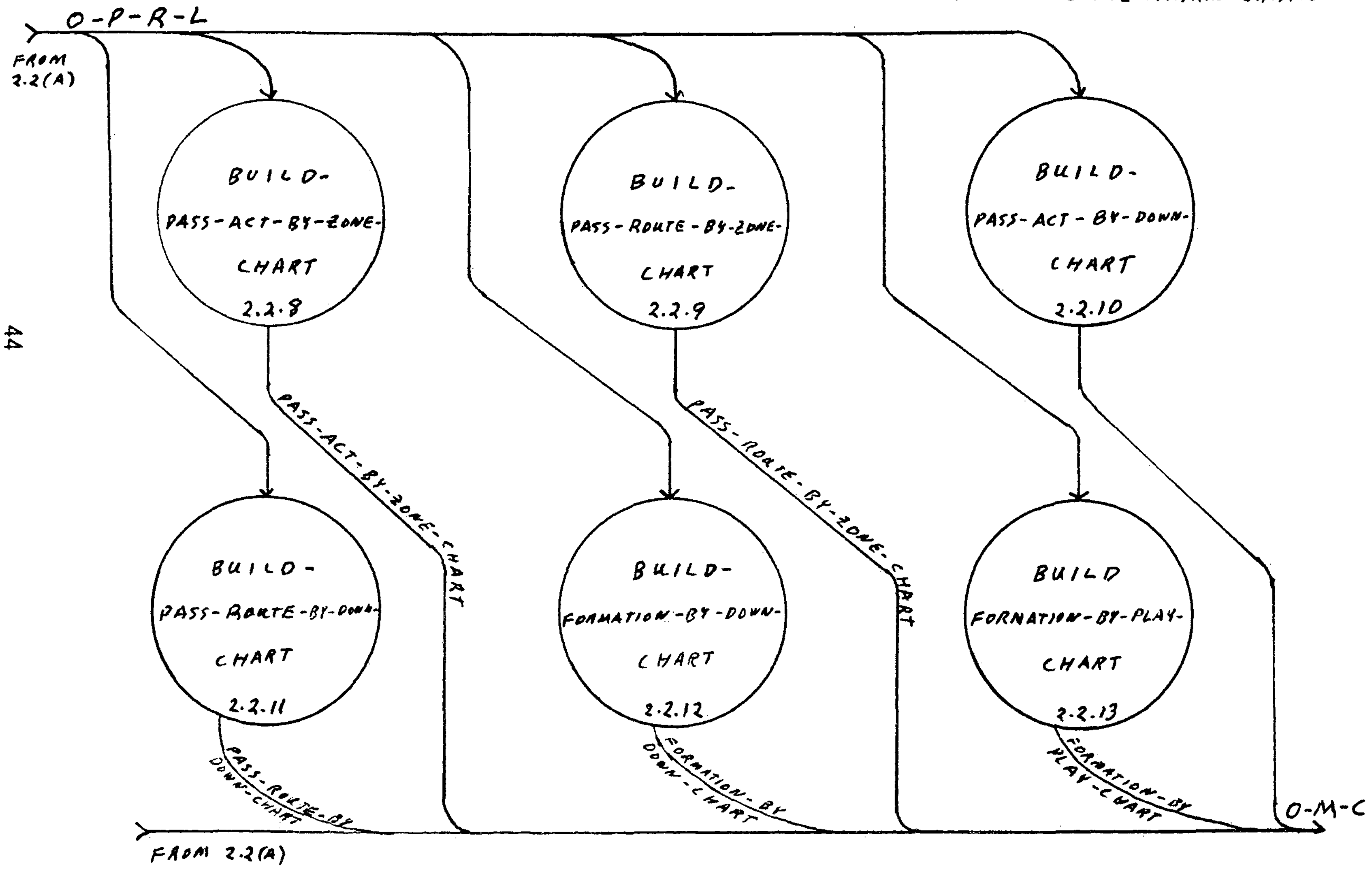

O-P-R-L
BUILD-HOLE-BY-DOWN-CHART 2.2.1
BUILD-HOLE-BY-HASH-CHART 2.2.2
TO 2.2(B)
BUILD-HOLE-BY-FORMATION-CHART 2.2.3
BUILD-DOWN-BY-PLAY-CHART 2.2.4
BUILD-PASS-ACT-BY-HASH-CHART 2.2.5
BUILD-PASS-ROUTE-BY-HASH-CHART 2.2.6
BUILD FORMATION-BY-HASH-CHART 2.2.7
HOLE-BY-DOWN-CHART
HOLE-BY-FORMATION-CHART
DOWN-BY-PLAY-CHART
PASS-ROUTE-BY-HASH-CHART
FORMATION-BY-HASH-CHART
HOLE-BY-HASH-CHART
PASS-ACT-BY-HASH-CHART
O-M-C
TO 2.2(B)

DIAGRAM 2.2(B)
BUILD-OFFENSIVE-MATRIX-CHARTS



O-P-R-L
FROM 2.2(A)
BUILD-PASS-ACT-BY-ZONE-CHART 2.2.8
BUILD-PASS-ROUTE-BY-ZONE-CHART 2.2.9
BUILD-PASS-ACT-BY-DOWN-CHART 2.2.10
BUILD-PASS-ROUTE-BY-DOWN-CHART 2.2.11
BUILD-FORMATION-BY-DOWN-CHART 2.2.12
BUILD FORNATION-BY-PLAY-CHART 2.2.13
PASS-ACT-BY-ZONE-CHART
PASS-ROUTE-BY-ZONE-CHART
PASS-ROUTE-BY DOWN-CHART
FORMATION-BY DOWN-CHART
FORMATION-BY PLAY-CHART
FROM 2.2(A)
O-M-C

DIAGRAM 2.3
BUILD-OFF-LIST-CHART

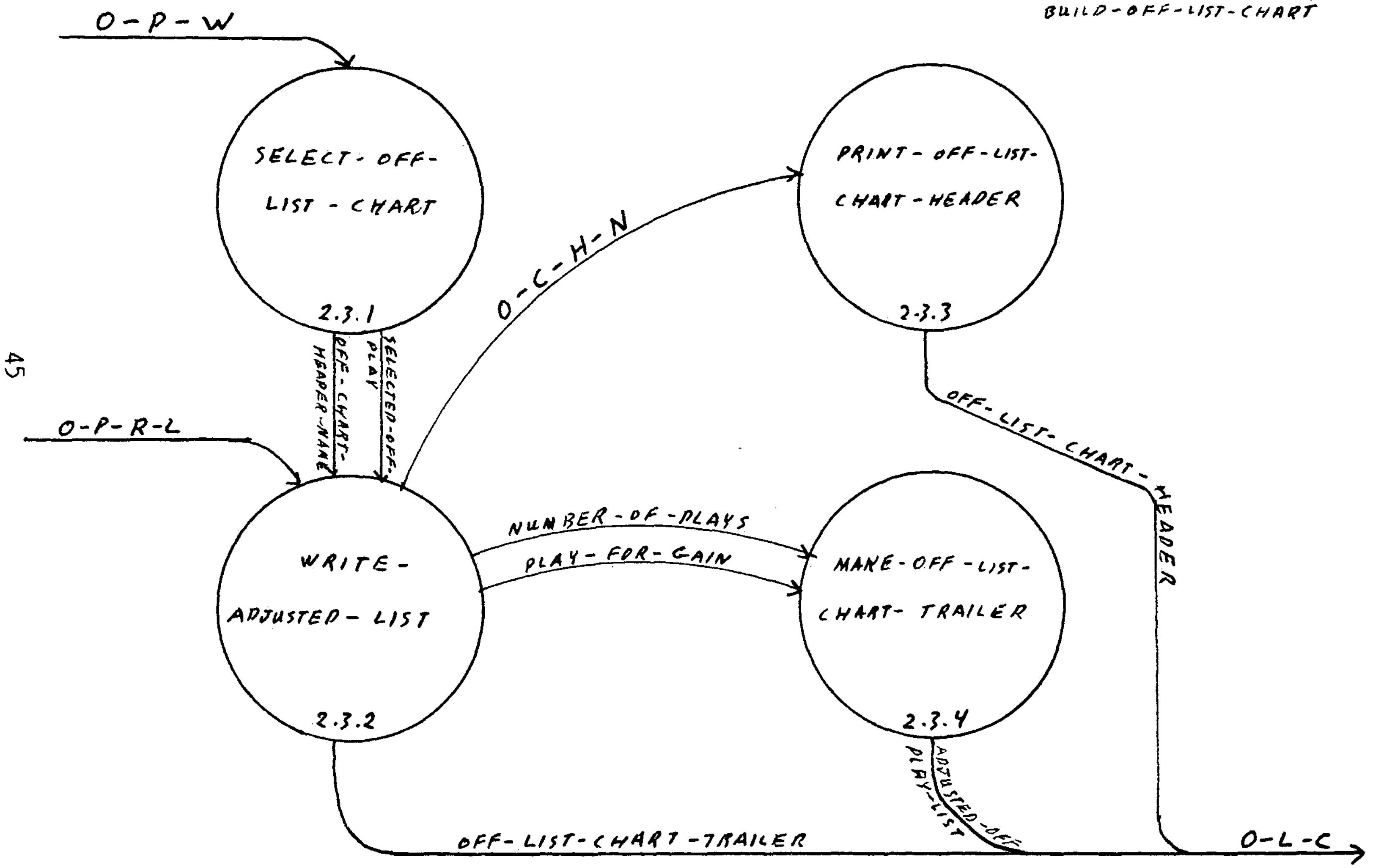
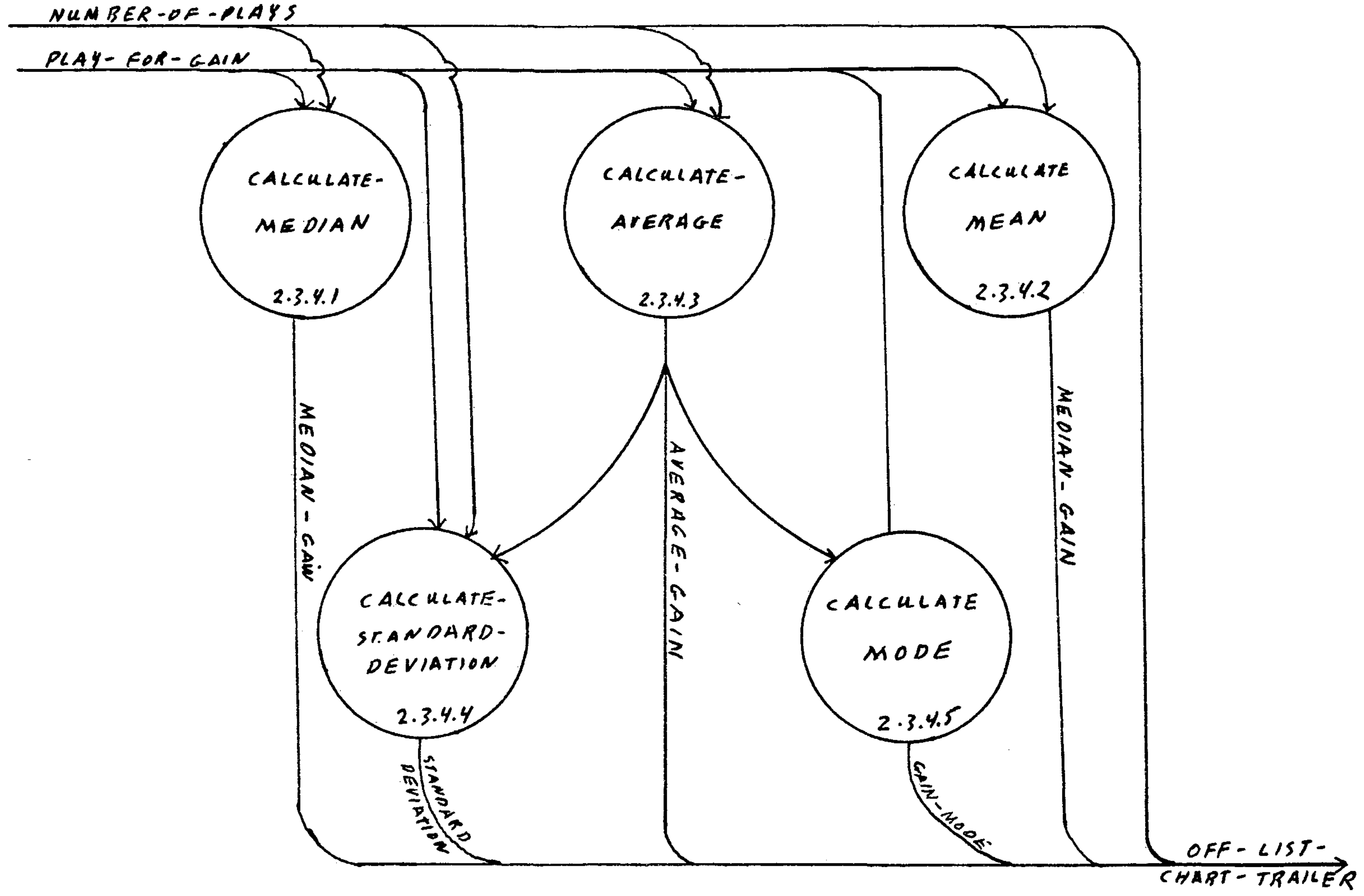


DIAGRAM 2.3.4
MAKE-OFF-LIST-CHART-TRAILER



DATA DICTIONARY

CONVENTIONS AND TERMS

Conventions:

1. {item} The item is repeated one or more times.

2. [item or item] One item or the other but not both.

3. and Both the preceding item and the following one.

4. and/or Both the preceding item and the following one or either one of them.

5. item : statement The item with the restrictions such that the statement is true.

6. or Exclusive or-- the preceding item or the following item but not both.

Terms:

1. Aliases This is a name used on the data flow charts that stands for the same data.

2. Composition This is the actual data items that are contained in the data flow.

3. Name This is the name of the data or process being described.

4. Process number This is the number of the circle on the data flow digrams where this process can be found.

DATA

NAME: Cover-by-down-chart
ALIASES: none
COMPOSITION: The number of times that each coverage is used for each down and distance.

NAME: Cover-by-formation-chart
ALIASES: none
COMPOSITION: The number of times that each coverage is used for each formation.

NAME: Cover-by-front-chart
ALIASES: none
COMPOSITION: The number of times that each coverage is used for each front.

NAME: Cover-by-hash-chart
ALIASES: none
COMPOSITION: The number of times that each coverage is used for each hash mark.

NAME: Cover-by-zone-chart
ALIASES: none
COMPOSITION: The number of times that each coverage is used for each yardage zone.

NAME: D-c-h-n
ALIASES: defensive-chart-heading-name
COMPOSITION: Alias for defensive-chart-heading-name

NAME: D-l-c
ALIASES: defensive-chart-list
COMPOSITION: Alias for defensive-chart-list

NAME: D-m-c
ALIASES: defensive-matrix-chart
COMPOSITION: Alais for defensive-matrix-chart

NAME: D-p-r-l
ALIASES: defensive-play-record-list
COMPOSITION: Alais for defensive-play-record-list

```
NAME: D-p-w
ALIASES: defensive-play-wanted
COMPOSITION:    alais for defensive-play-wanted

NAME: Defensive-chart-heading
ALIASES: none
COMPOSITION:    defensive-chart-header-name    and
                the headings for the columns on the
                defensive list charts.

NAME: Defensive-chart-heading-name
ALIASES: d-c-h-n
COMPOSITION:    The name of the list chart wanted

NAME: Defensive-list-chart
ALIASES: d-l-c
COMPOSITION:    Defensive-chart-heading       and
                Selected-play-record-list

NAME: Defensive-matrix-chart
ALIASES: d-m-c
COMPOSITION:    cover-by-front-chart       or
                cover-by-formation-chart   or
                cover-by-zone-chart        or
                cover-by-hash-chart        or
                cover-by-down-chart        or
                front-by-formation-chart   or
                front-by-down-chart        or
                front-by-zone-chart        or
                front-by-hash-chart

NAME: Defensive-play
ALIASES: none
COMPOSITION:    opponent              and
                hash mark             and
                yardage zone          and
                down and distance     and
                offensive formation   and
                motion                and
                front                 and
                variation to front    and
                coverage              and
                stunt                 and
                blitz

NAME: Defensive-play-file
ALIASES: none
COMPOSITION:   {defensive-play}
```

```
NAME: Defensive-play-record-list
ALIASES: d-p-r-l
COMPOSITION:    {defensive play}

NAME: Defensive-play-wanted
ALIASES: d-p-w,selected-defensive-play
COMPOSITION:   [opponent or not-specified]             and
               [hash mark or not-specified]            and
               [yardage zone or not-specified]         and
               [down and distance or not-specified]    and
               [offensive formation or not-specified]  and
               [motion or not-specified]               and
               [front or not-specified]                and
               [variation to front or not-specified]   and
               [coverage or not-specified]             and
               [stunt or not-specified]                and
               [blitz or not-specified]

NAME: Front-by-down-chart
ALIASES: none
COMPOSITION:    The number of times that each front
                is used for each down and distance.

NAME: Front-by-formation-chart
ALIASES: none
COMPOSITION:    The number of times that each front
                is used for each formation.

NAME: Front-by-hash-chart
ALIASES: none
COMPOSITION:    The number of times that each front
                is used for each hash mark.

NAME: Front-by-zone-chart
ALIASES: none
COMPOSITION:    The number of times that each front
                is used for each yardage zone.

NAME: Selected-defensive-play
ALIASES: d-p-w,defensive-play-wanted
COMPOSITION:    Alias for defensiv-play-wanted

NAME: Selected-play-record-list
ALIASES: none
COMPOSITION:    {selected defensive play}
  (Comment: a selected defensive play is a defensive
            play in which selected fields are
            restricted to selected values)
```

PROCESSES

```
NAME: Read-new-def-plays
PROCESS NUMBER:  1.1.1
DESCRIPTION:    1. If new plays are to be added then
                  A. While all new plays have not been
                     read.
                    a. Read each item of a defensive play
                       from the user.

NAME: Read-old-def-plays
PROCESS NUMBER:  1.1.2
DESCRIPTION:    1. If old plays are to be included then
DESCRIPTION:      A. While all old plays have not been
                     read.
                    a. Read each item of a defensive play
                       from the user.

NAME: Build def-play-record-list
PROCESS NUMBER:  1.1.3
DESCRIPTION:    1. For each play read in put the play
                   into a list of defensive plays.

NAME: Build-cover-by-front-chart
PROCESS NUMBER:  1.2.1
DESCRIPTION:    1. For each play in the defensive-play-
                   record-list
                   A. Increment the element of the
                      cover by front array that is
                      indexed by the cover and front.
                2. Output the cover by front array.

NAME: Build-cover-by-formation-chart
PROCESS NUMBER:  1.2.2
DESCRIPTION:    1. For each play in the defensive-play-
                   record-list
                   A. Increment the element of the
                      cover by formation array that is
                      indexed by the cover and formation.
                2. Output the cover by formation array.
```

```
NAME: Build-cover-by-zone-chart
PROCESS NUMBER:  1.2.3
DESCRIPTION:     1. For each play in the defensive-play-
                    record-list
                    A. Increment the element of the
                       cover by zone array that is
                       indexed by the cover and zone.
                 2. Output the cover by zone array.

NAME: Build-cover-by-hash-chart
PROCESS NUMBER:  1.2.4
DESCRIPTION:     1. For each play in the defensive-play-
                    record-list
                    A. Increment the element of the
                       cover by hash array that is
                       indexed by the cover and hash.
                 2. Output the cover by hash array.

NAME: Build-cover-by-down-chart
PROCESS NUMBER:  1.2.5
DESCRIPTION:     1. For each play in the defensive-play-
                    record-list
                    A. Increment the element of the
                       cover by down array that is
                       indexed by the cover and down.
                 2. Output the cover by down array.

NAME: Build-front-by-formation-chart
PROCESS NUMBER:  1.2.6
DESCRIPTION:     1. For each play in the defensive-play-
                    record-list
                    A. Increment the element of the
                       front by formation array that is
                       indexed by the front and formation.
                 2. Output the front by formation array.

NAME: Build-front-by-down-chart
PROCESS NUMBER:  1.2.7
DESCRIPTION:     1. For each play in the defensive-play-
                    record-list
                    A. Increment the element of the
                       front by down array that is
                       indexed by the front and down.
                 2. Output the front by down array.
```

```
NAME: Build-front-by-zone-chart
PROCESS NUMBER:  1.2.8
DESCRIPTION:     1. For each play in the defensive-play-
                    record-list
                    A. Increment the element of the
                       front by zone array that is
                       indexed by the front and zone.
                 2. Output the front by zone array.

NAME: Build-front-by-hash-chart
PROCESS NUMBER:  1.2.9
DESCRIPTION:     1. For each play in the defensive-play-
                    record-list
                    A. Increment the element of the
                       front by hash array that is
                       indexed by the front and hash.
                 2. Output the front by hash array.

NAME: Select-def-list-chart
PROCESS NUMBER:  1.3.1
DESCRIPTION:     1  Get the defensive-wanted-play.
                 2. Assign the name of the defensive-
                    list chart.

NAME: Print-def-list-chart-header
PROCESS NUMBER:  1.3.2
DESCRIPTION:     1. Output the name of the chart.
                 2. Output the column headings for
                    the defensive-list-chart.

NAME: Write-adjusted-list
PROCESS NUMBER:  1.3.3
DESCRIPTION:     1. For each record from the
                     defensive-play-record-list.
                   A. If for each field
                     a. Defensive-play-wanted field
                         is not specified.
                      or
                     b. Defensive-play-wanted field =
                         defensive-play-record field.
                      then output defensive-play-record.
```

DATA DICTIONARY
o-SCOUT

Data

```
NAME: Adjusted-off-play-list
ALIASES: none
COMPOSITION:    {selected offensive play}
   (comment: an adjusted offensive play is an
             offensive play where selected fields
             are restricted to selected values)

NAME: average-gain
ALIASES: none
COMPOSITION:     The average of the yards gained or
                 lost by the plays on the
                 offensive-list-chart.

NAME: Down-by-play-chart
ALIASES: none
COMPOSITION:    The number of times that each play is used
                for each down and distance.

NAME: formation-by-down-chart
ALIASES: none
COMPOSITION:    The number of times that each formation is
                used for each down and distance.

NAME: formation-by-hash-chart
ALIASES: none
COMPOSITION:    The number of times that each formation is
                used for each hash mark.

NAME: formation-by-play-chart
ALIASES: none
COMPOSITION:    The number of times that each formation is
                used for each play.

NAME: gain-mode
ALIASES: none
COMPOSITION:     The mode of the yards gained or
                 lost by the plays on the
                 offensive-list-chart.
```

NAME: Hole-by-down-chart
ALIASES: none
COMPOSITION: The number of times that each hole is used for each down and distance.

NAME: Hole-by-formation-chart
ALIASES: none
COMPOSITION: The number of times that each formation is used for each down and distance.

NAME: hole-by-hash-chart
ALIASES: none
COMPOSITION: The number of times that each hole is used for each hash mark.

NAME: Mean-gain
ALIASES: none
COMPOSITION: The mean of the yards gained or lost by the plays on the offensive-list-chart.

NAME: Median-gain
ALIASES: none
COMPOSITION: The median of the yards gained or lost by the plays on the offensive-list-chart.

NAME: Number-of-plays
ALIASES: none
COMPOSITION: the number of plays on the offensive-list-chart.

NAME: O-c-h-n
ALIASES: offensive-chart-header-name
COMPOSITION: Alias for offensive-chart-header-name

NAME: O-l-c
ALIASES: offensive-list-chart
COMPOSITION: Alias for offensive-liat-chart

NAME: O-m-c
ALIASES: offensive-matrix-chart
COMPOSITION: Alias for offensive-matrix-chart

NAME: O-p-r
ALIASES: offensive-play-record
COMPOSITION: Alias for offensive-play-record

```
NAME: O-p-w
ALIASES: offensive-play-wanted,selected-offensive-play
COMPOSITION:    Alias for offensive-play-wanted

NAME: Offensive-chart-header-name
ALIASES: o-c-h-n
COMPOSITION:    The name of the list chart wanted.

NAME: Offensive-list-chart
ALIASES: o-l-c
COMPOSITION:    offensive-chart-heading       and
                adjusted-offensive-play-list  and
                offensive-list-chart-trailer

NAME: Offensive-chart-heading
ALIASES: none
COMPOSITION:    offensive-chart-header-name   and
                the column headings for the plays
                on the offensive-list-chart.

NAME: Offensive-list-chart-trailer
ALIASES: none
COMPOSITION:    number-of-plays            and
                average-gain               and
                median-gain                and
                mean-gain                  and
                gain-mode                  and
                standard-deviation

NAME: Offensive-matrix-chart
ALIASES: o-m-c
COMPOSITION:    hole-by-down-chart         or
                hole-by-hash-chart         or
                hole-by-formation-chart    or
                down-by-play-chart         or
                pass-act-by-hash-chart     or
                pass-route-by-hash-chart   or
                formation-by-hash-chart    or
                pass-act-by-zone-chart     or
                pass-route-by-zone-chart   or
                pass-act-by-down-chart     or
                pass-route-by-down-chart   or
                formation-by-down-chart    or
                formation-by-play-chart.
```

```
NAME: Offensive-play
ALIASES: none
COMPOSITION:    o-opponent                  and
                o-hash mark                 and
                o-yardage zone              and
                o-down and distance         and
                o-formation                 and
                run or pass                 and
                play ran                    and
                ball carriers position      and
                ball carriers number        and
                hole number                 and
                strong or weak side         and
                gain or loss                and
                pass action                 and
                pass result                 and
                receivers number            and
                receivers position          and
                pass pattern                and
                pass depth                  and
                pass zone                   and
                drive number                and
                play number

NAME: Offensive-play-file
ALIASES: none
COMPOSITION:     {offensive-play}

NAME: Offensive-play-wanted
ALIASES: o-p-w,selected-off-play
COMPOSITION:  [o-opponent or not-selected]           and
              [o-hash mark or not-selected]          and
              [o-yardage zone or not-selected]       and
              [o-down and distance or not-selected]  and
              [o-formation or not-selected]          and
              [run or pass or not-selected]          and
              [ball carriers position or not-selected] and
              [ball carriers number or not-selected] and
              [hole number or not-selected]          and
              [strong or weak side or not-selected]  and
              [gain or loss or not-selected]         and
              [pass action or not-selected]          and
              [pass result or not-selected]          and
              [receivers number or not-selected]     and
              [receivers position or not-selected]   and
              [pass pattern or not-selected]         and
              [pass depth or not-selected]           and
              [pass zone or not-selected]            and
```

```
                [drive number or not-selected]          and
                [play number or not-selected]
          (comment: not-selected means that this is not one
                    of the keys that the user wants
                    considered for this chart.)

NAME: Offensive-play-record
ALIASES: o-p-r
COMPOSITION:     {offensive-play}

NAME: pass-act-by-down-chart
ALIASES: none
COMPOSITION:    The number of times that each pass action
                is used for each down.

NAME: pass-act-by-hash-chart
ALIASES: none
COMPOSITION:    The number of times that each pass action
                is used for each hash mark.

NAME: pass-act-by-zone-chart
ALIASES: none
COMPOSITION:    The number of times that each pass action
                is used for each yardage zone.

NAME: pass-route-by-down-chart
ALIASES: none
COMPOSITION:    The number of times that each pass route
                is used for each down and distance.

NAME: pass-route-by-hash-chart
ALIASES: none
COMPOSITION:    The number of times that each pass route
                is used for each hash mark.

NAME: pass-route-by-zone-chart
ALIASES: none
COMPOSITION:    The number of times that each pass route
                is used for each yardage zone.

NAME: play-for-gain
ALIASES: none
COMPOSITION:    The number of offensive plays that
                have each gain or loss value.

NAME: Selected-offensive-play
ALIASES: o-p-w,offensive-play-wanted
COMPOSITION:    Alias for offensive-play-wanted
```

NAME: standard-deviation
ALIASES: none
COMPOSITION: The standard deviation for the gain or loss value of the plays on the offensive-list-chart.

PROCESSES

NAME: Read-new-off-plays
PROCESS NUMBER: 2.1.1
DESCRIPTION:

```
1. If new plays are to be included then
  A. While all new plays have not been read
    a. Read the items of the new play form
       the user.
```

NAME: Read-old-off-plays
PROCESS NUMBER: 2.1.2
DESCRIPTION:

```
1. If old plays are to be included then
  A  While all old plays have not been read
    a. Read the items of the old play from
       the offensive-play-file.
```

NAME: Build-off-play-record-list
PROCESS NUMBER: 2.1.3
DESCRIPTION:

```
1. For each play read in add it to a
   linked list in the order that the
   plays are read in.
```

NAME: Build-sorted-off-play-record-list
PROCESS NUMBER: 2.1.4
DESCRIPTION:

```
1. For each play read in add it to a
   linked list in ascending order by
   the number of yards gained or lost.
```

NAME: Build-hole-by-down-chart
PROCESS NUMBER: 2.2.1
DESCRIPTION:

```
1. For each play in the defensive-play-
   record-list.
  A. increment the element of the hole
     by down array indexed by the
     hole and down.
2. Output the hole by down array.
```

```
NAME: Build-hole-by-hash-chart
PROCESS NUMBER:  2.2.2
DESCRIPTION:     1. For each play in the defensive-play-
                    record-list.
                   A. increment the element of the hole
                      by hash array indexed by the
                      hole and hash.
                 2. Output the hole by hash array.

NAME: Build-hole-by-formation-chart
PROCESS NUMBER:  2.2.3
DESCRIPTION:     1. For each play in the defensive-play-
                    record-list.
                   A. increment the element of the hole
                      by formation array indexed by the
                      hole and formation.
                 2. Output the hole by formation array.

NAME: Build-down-by-play-chart
PROCESS NUMBER:  2.2.4
DESCRIPTION:     1. For each play in the defensive-play-
                    record-list.
                   A. increment the element of the play
                      by formation array indexed by the
                      play and formation.
                 2. Output the play by formation array.

NAME: Build-pass-act-by-hash-chart
PROCESS NUMBER:  2.2.5
DESCRIPTION:     1. For each play in the defensive-play-
                    record-list.
                   A. increment the element of the pass
                      action by hash array indexed by
                      the pass action and hash mark.
                 2. Output the pass action by hash array.

NAME: Build-pass-route-by-hash-chart
PROCESS NUMBER:  2.2.6
DESCRIPTION:     1. For each play in the defensive-play-
                    record-list.
                   A. increment the element of the pass
                      route by hash array indexed by
                      the pass route and hash mark.
                 2. Output the pass route by hash array.
```

NAME: Build-formation-by-hash-chart
PROCESS NUMBER: 2.2.7
DESCRIPTION:

```
1. For each play in the defensive-play-
   record-list.
  A. increment the element of the
     formation by hash array indexed by
     the formation and hash mark.
2. Output the  formation by hash array.
```

NAME: Build-pass-act-by-zone-chart
PROCESS NUMBER: 2.2.8
DESCRIPTION:

```
1. For each play in the defensive-play-
   record-list.
  A. increment the element of the pass
     action by zone array indexed by
     the action and zone.
2. Output the pass action by zone array.
```

NAME: Build-pass-route-by-zone-chart
PROCESS NUMBER: 2.2.9
DESCRIPTION:

```
1. For each play in the defensive-play-
   record-list.
  A. increment the element of the pass
     route by zone array indexed by
     the pass route and zone.
2. Output the pass route by zone array.
```

NAME: Build-pass-act-by-down-chart
PROCESS NUMBER: 2.2.10
DESCRIPTION:

```
1. For each play in the defensive-play-
   record-list.
  A. increment the element of the pass
     action by down array indexed by
     the pass action and down.
2. Output the pass action by down array.
```

NAME: Build-pass-route-by-down-chart
PROCESS NUMBER: 2.2.11
DESCRIPTION:

```
1. For each play in the defensive-play-
   record-list.
  A. increment the element of the pass
     route by down array indexed by
     the pass route and down.
2. Output the pass route by down array.
```

```
NAME: Build-formation-by-down-chart
PROCESS NUMBER:  2.2.12
DESCRIPTION:     1. For each play in the defensive-play-
                    record-list.
                   A. increment the element of the
                      formation by down array indexed by
                      the formation and down.
                 2. Output the formation by down array.

NAME: Build-formation-by-play-chart
PROCESS NUMBER:  2.2.13
DESCRIPTION:     1. For each play in the defensive-play-
                    record-list.
                   A. increment the element of the
                      formation by play array indexed by
                      the formation and play.
                 2. Output the formation by play array.

NAME: Select-off-list-chart
PROCESS NUMBER:  2.3.1
DESCRIPTION:     1  Get the selected-off-play.
                 2. Assign the name of the offensive-
                    list chart.

NAME: Print-off-list-chart-header
PROCESS NUMBER:  2.3.2
DESCRIPTION:     1. Output the name of the chart.
                 2. Output the column headings for
                    the offensive-list-chart.

NAME: Write-adjusted-list
PROCESS NUMBER:  2.3.3
DESCRIPTION:     1. For each record from the
                     offensive-play-record-list.
                   A. If for each field
                     a. Offensive-play-wanted field
                         is not specified.
                      or
                     b. Offensive-play-wanted field =
                         offensive-play-record field.
                     then output offensive-play-record.
```

```
NAME: Calculate-median
PROCESS NUMBER:  2.3.4.1
DESCRIPTION:     1. Beginning with the most yards lost
                    or the least yards gained if no yards
                    lost and counting each play toward the
                    largest gain select the play that
                    is half of the number-of plays and
                    return the gain of this play.

NAME: Calculate-mean
PROCESS NUMBER:  2.3.4.2
DESCRIPTION:     1. Determine the play with the largest
                    gain and the one with the largest
                    loss.
                 2. Find the average gain or loss if
                    the play with the largest gain and
                    the play with the largest loss are
                    not counted.

NAME: Calculate-average
PROCESS NUMBER:  2.3.4.3
DESCRIPTION:     1. Find the average gain for all the
                    plays on the chart.

NAME: Calculate-standard-deviation
PROCESS NUMBER:  2.3.4.4
DESCRIPTION:     1. Find the standard deviation for the
                    gain and loss values of the plays
                    on the offensive-list-chart.

NAME: Calculate-mode
PROCESS NUMBER:  2.3.4.5
DESCRIPTION:     1. Find the gain or loss of the middle
                    play on the offensive-list-chart.
```

CHART O
d-SCOUT

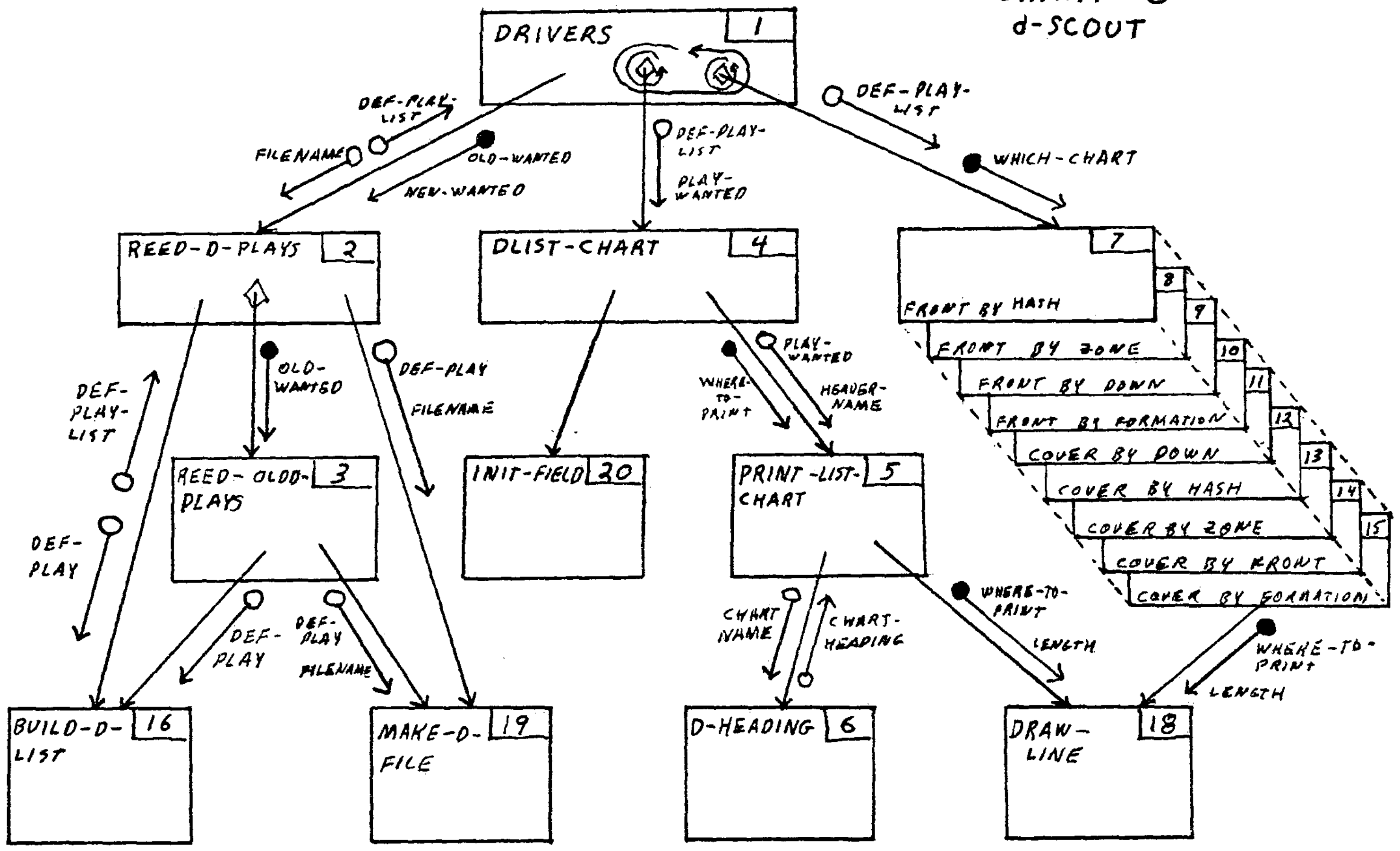


CHART 1
DRIVERS

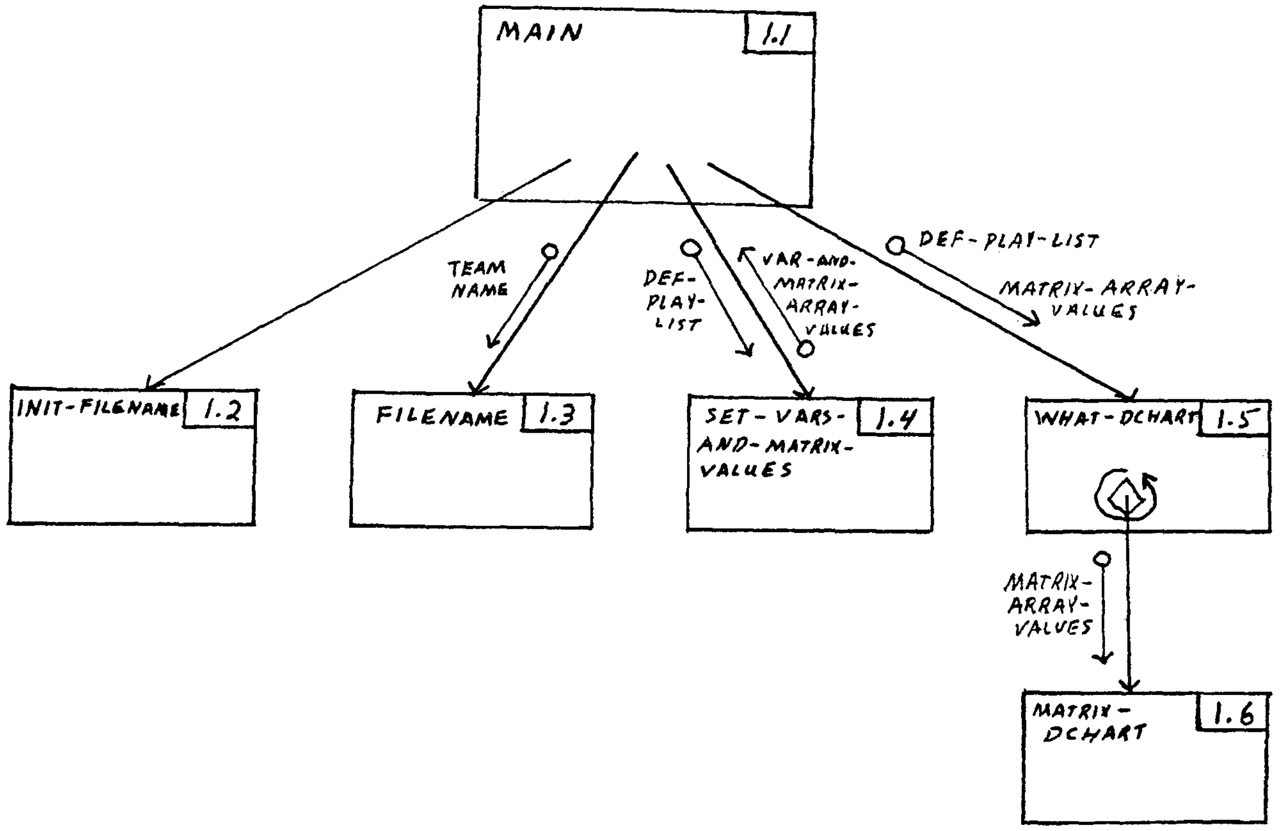


CHART 1.4
SET-VARS-AND MATRIX-VALUES

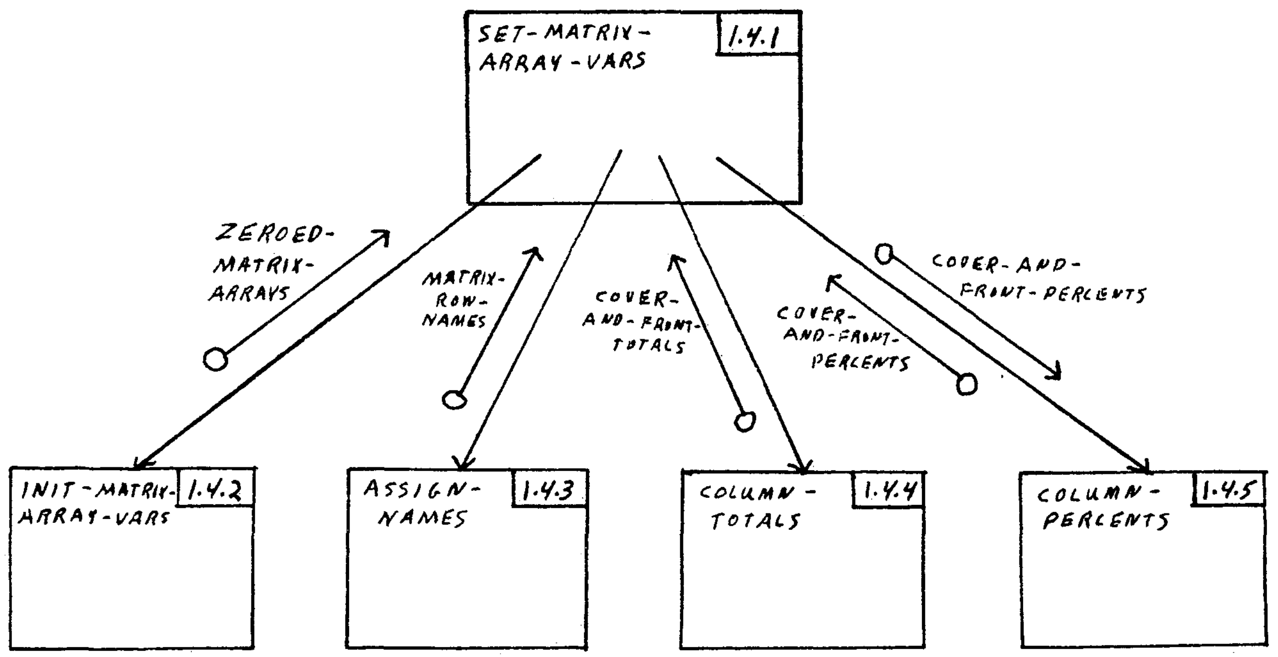
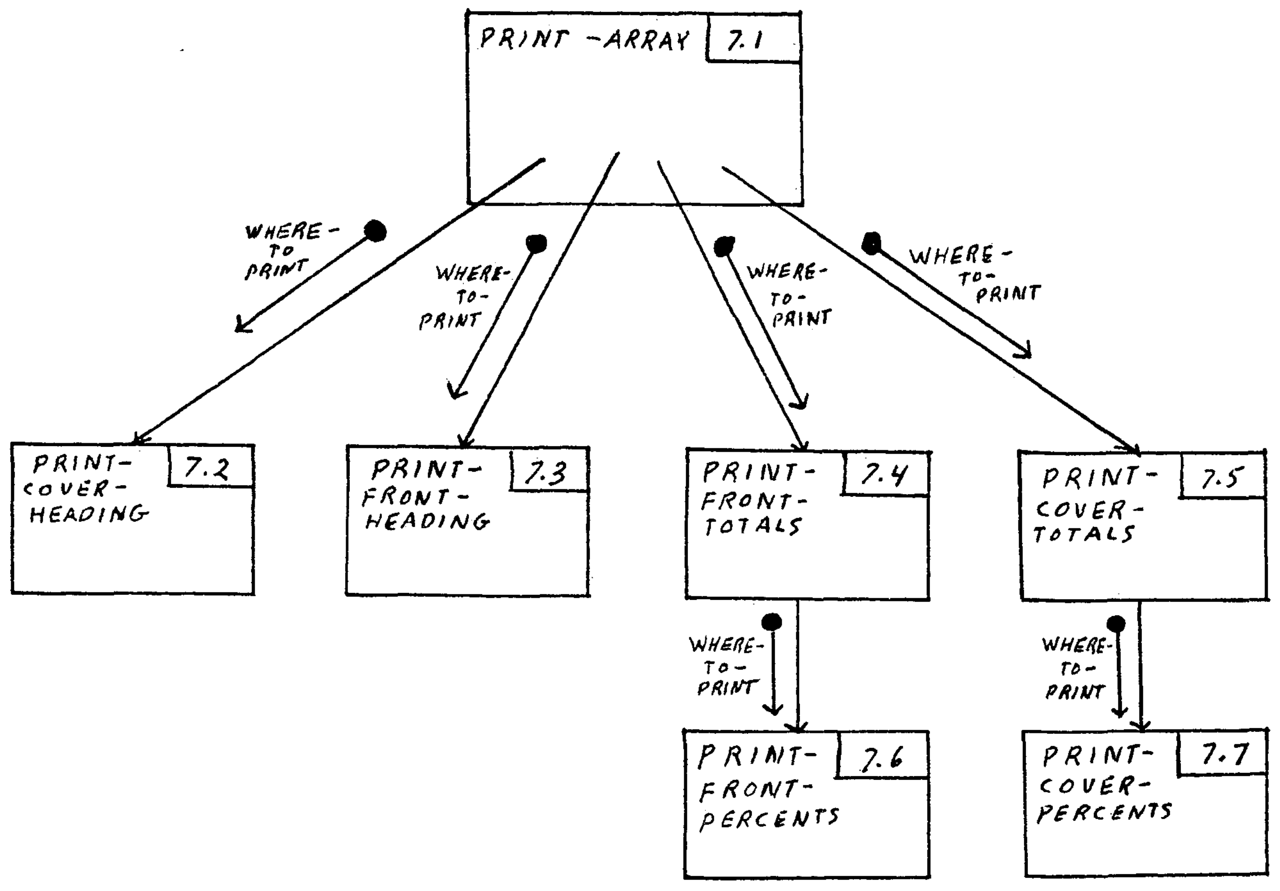


CHART 7-15
PRINT-MATRIX-CHART-ARRAYS



DESIGN DOCUMENTATION FOR SCOUT

-by- K. Garry Dyer

Scout was orininally written as one program but it grew to a size such that the facilities at the U. of M. would not run it so I divided it into two programs, d-SCOUT for defense, and o-SCOUT for offense.

The design documentation for them is separated here and even though some of the procedures are very similar in both programs I have included them in both.

d-SCOUT

CONSTANTS

These constants are the names that the University of Montana football teams use and the program can be adopted for any coaching staff by changing these names and the headings on the output charts.

```
    front1 = 'OKIE    '    front9 = '42      '
    front2 = 'GRIZ    '    front10 = '42SK    '
    front3 = 'SLST    '    front11 = '43SK    '
    front4 = 'SLWK    '    front12 = '44SK    '
    front5 = 'SLTE    '    front13 = '61      '
    front6 = '43      '    front14 = '62      '
    front7 = 'P43     '    front15 = 'EST     '
    front8 = 'C43     '    front16 = 'EWK     '
    form1 = 'SPLA    '     form10 = 'PRS     '
    form2 = 'SPRA    '     form11 = 'SLI     '
    form3 = 'SPLB    '     form12 = 'SRI     '
    form4 = 'SPRB    '     form13 = 'SLQ     '
    form5 = 'PRI     '     form14 = 'SRQ     '
    form6 = 'PLI     '     form15 = 'SLS     '
    form7 = 'PLQ     '     form16 = 'SRS     '
    form8 = 'PRQ     '     form17 = 'TWRI    '
    form9 = 'PLS     '     form18 = 'TWLI    '
    cover1 = '1       '    cover8 = 'M2D     '
    cover2 = '2       '    cover9 = 'WMM     '
    cover3 = '3       '    cover10 = 'WM2     '
    cover4 = '5       '    cover11 = 'WM3     '
    cover5 = '6       '    cover12 = 'COM     '
    cover6 = 'M       '    cover13 = 'ROB     '
    cover7 = 'MF      '
```

TYPES

```
COVERS = (one,two,three,five,six,M,MF,M2D,WMM,
          WM2,WM3,COM,ROB,othercover)
FRONTS = (OKIE,GRIZ,SLST,SLWK,SLTE,F43,P43,C43,
          F42,F42SK,F43SK,F44SK,F61,F62,EST,
          EWK,otherfront)
FORMATIONS = (SPLA,SPRA,SPLB,SPRB,PRI,PLI,PLQ,
              PRQ,PLS,PRS,SLI,SRI,SLQ,SRQ,SLS,
              SRS,TWRI,TWLI,otherform)
ZONES = (zone,ztwo,zthree,zfour,zfive,zsix,
         otherzone)
HASHMARKS = (RIGHT,MIDDLE,LEFT,otherhash)
DOWNS = (OL,OM,OS,OG,SL,SM,SS,SG,TL,TM,TS,
         TG,FL,FM,FS,FG,otherdown)
D_PTR a pointer to a record of defensive plays
TEAM_NAME a packed array of 15 elements to be
  used to name files.
TWO_STRING a packed array of two elements
THREE_STRING a packed array of three elements
FOUR_STRING a packed array of four elements
FIVE_STRING a packed array of five elements
EIGHT_STRING a packed array of eight elements
TWELVE_STRING a packed array of twelve elements

D_PLAY_RECORD a record of the defensive plays containing
   the following fields:
   OT of type three_string to hold the opponent
   HASH of type char to hold the hash mark
   ZONE of type char to hold the yardage zone
   D_D of type two_string to hold the down
      and distance
   FORM of type eight_string to hold the
      offensive formation
   MOTION of type three_string to hold any
      motion the offense uses.
   FRONT of type eight_string to hold the
      front used by the defense
   VARS of type three_string to hold any
      variation to the front
   COVER of type eight_string to hold the
      secondary coverage
   STUNT of type eight_string to hold the
      stunts
   BLITZ of type eight_string to hold the
      blitzes
```

NEXT_DP of type d_ptr to point to the next
   record in the linked list of defensive
   plays

GLOBAL VARIABLES

COVER_TOTALS a packed array of 14 elements of type integer to hole the totals for the coverage charts.
FRONTNAMES an array of 17 elements of type four_string to hold the names of the fronts.
COVER_BY_FRONT_AR a two dimensional array of covers by fronts of type integer to hold the number of times each coverage is used with each front.
FORMATIONNAMES an array of 19 elements of type four_string to hold the names of the formations.
COVER_BY_FORMATION_AR a two dimensional array of covers by formations of type integer to hold how many times each coverage is used for each formation.
COVER_BY_ZONE_AR a two dimensional array of covers by zones of type integer to hold how many times each coverage is used for each zone.
COVER_BY_HASH_AR a two dimensional array of covers by hashmarks of type integer to hold how many times each coverage is used for each hash mark.
HASHNAMES an array of 7 elements of type four_string to hold the names of the hash marks.
DOWNNAMES an array of 17 elements of type four_string to hold the names of the downs.
COVER_BY_DOWN_AR a two dimensional array of covers by downs of type integer to hold how many times each coverage is used for each down.
FRONT_BY_FORMATION_AR a two dimensional array of fronts by formations of type integer to hold how many times each front is used for each formation.
FRONT_BY_DOWN_AR a two dimensional array of fronts by downs of type integer to hold how many times each front is used for each down.
FRONT_BY_ZONE_AR a two dimensional array of fronts by zones of type integer to hold how many times each front is used for each zone.
FRONT_TOTALS an array of 17 elements of type integer to hold the total number of times that each front is used.
FRONT_BY_HASH_AR a two dimensional array of fronts by hashmarks of type integer to hold how many times each front is used for each hash mark.
ZONENAMES an array of 7 elements of type four_string to hold the names of the zones.

INFILE,OUTFILE of type text
D_ANCHOR of type d_ptr to point to the first
  defensive record
FILE_INO,FILE_IND,FILE_OUT1,FILE_OUT2,
FILE_OUT3,FILE_OUT4 of type team_name
  to hold the names of the input and output files
ALLDONE of type char
D_RECORD of type d_play_record, a record of the
  defensive play
DCURRENT,DTEMP_PTR of type d_ptr
DHASH of type char to hold the hash mark
DZONE to hold the yardage zone
DD_D of type two_string to hold the down and
   distance
DFORM of type eight_string to hold the
  formation of the play
DOT of type three_string to hold the opponent
DFRONT of type eight_string to hold the defensive
  front
DVAR of type three_string to hold any variation
  to the front
DCOVER of type eight_string to hold the secondary
  coverage
DSTUNT of type eight_string to hold any defensive
  stunts
DBLITZ of type eight_string to hold any blitzes
DMOTION of type three_string to hold any motion
  used by the offense.
DC of type char to hold if charts are wanted.
TOTAL_DPLAYS of type integer to hold the total
  number of plays entered.
D_HEADED of type boolean to tell if a chart
  has a heading
WRITE_TO_DFILE of type boolean to tell if the
  chart is to a file or to the terminal
FIELD a packed array of twenty elements
NUMPLAYS of type integer
ODD_DKEY of type boolean
NUMDKEYS of type integer
MANY_DKEYS of type boolean
FIRSTD of type boolean
DREQUEST of type five_string
WHICH_DKEY of type integer
DFILE_SET of type boolean to tell if the
  outfile has been set to the proper name
MMENU,LMENU of type boolean to tell if the
  user has seen the menus.
COVERPERCENTS an array of 15 elements of type

  integer to hole the percentage of times that
  each coverage is used.
FRONTPERCENTS an array of 18 elements of type
  integer to hole the percentage of times that
  each front is used.

PROCEDURES

1.1 MAIN

This is the main driver for the rest of the program.

ALGORITHM

```
  1. Repeat until the user is finished with the
     program.    (all_done = 'n')
    A. Assign false to and dfile_set.
    B. Write d-scout in large letters on lines
     10 through 15.
    C. Assign nil to d_anchor,dtemp_ptr.
    D. Assign nil to t1_danchor,t12_danchor.
    E. Assign false to o_headed, and d_headed.
    F. Assign 0 to total_dplays.
    G. Call procedure init_filename.
    H. Reset the input to the terminal.
    I. Ask the user what team is being scouted.
    J. Assign his answer to file_out1.
    K. Assign file_out1 to file_ino.
    L. Call procedure filename.
    N. Call procedure reed_d_plays.
    M. Ask the user if he wants defensive charts.
    O. If he does call procedure what_dchart.
    P. Ask the user if he wants to run the program
       again.
    Q. If he is finished then
      a. Assign N to alldone.
```

1.2 INIT_FILENAME

This procedure initializes the file name variables to spaces at the start of each run. It is called from main.

Local variable
  i of type integer.

ALGORITHM
1. Assign space to each of the elements of file_out1
2. Assign file_out1 to file_out2.
3. Assign file_out1 to file_out3.
4. Assign file_out1 to file_out4.
5. Assign file_out1 to file_ino.
6. Assign file_out1 to file_ind.

1.3 FILENAME

This procedure assigns the proper names to the variables to be used as file names by the program. It is called from main.

```
Local variable
 i of type integer

ALGORITHM
  1. Assign file_out1 to file_out2.

  2. Assign file_out1 to file_out3.

  3. Assign file_out1 to file_out4.

  4. Assign 1 to i.

  5. Repeat until file_out1 indexed by i is a space
     A. Increment i.

  6. Assign a period to file_out1 indexed by i.

  7. Assign a period to file_out2 indexed by i.

  8. Assign a period to file_out3 indexed by i.

  9. Assign a period to file_out4 indexed by i.

 10. Assign a O to file_out1 indexed by i+1.

 11. Assign a D to file_out2 indexed by i+1.

 12. Assign a O to file_out3 indexed by i+1.

 13. Assign a D to file_out4 indexed by i+1.

 14. Assign a C to file_out1 indexed by i+2.

 15. Assign a C to file_out1 indexed by i+2.

 16. Assign each element of file_out1 to file_ino.

 17. Assign each element of file_out2 to file_ind.
```

1.4.1 SET_MATRIX_ARRAY_VARS

This procedure is used to put the values into the elements of the matrix array variables.

```
Local variables
  tempptr of type d_ptr
  f of type fronts
  h of type hashmarks
  z of type zones
  d of type downs
  fo of type formations
  c of type covers

ALGORITHM
  1. Assign the pointer to the first play
     in the linked list of plays to tempptr.

  2. For each play in the linked list of plays do
    A. Assign otherfront to f.
    B. Assign otherhash to h.
    C. Assign otherzone to z.
    D. Assign otherdown to d.
    E. Assign otherform to fo.
    F. Assign othercover to c.
    G. Check the necassary fields of the play
       that tempptr points to and
      a. If the front field is front(n) assign
         the nth element to fronts to f.
      b. If the hash field is hash(n) assign
         the nth element to hashmarks to h.
      a. If the zone field is zone(n) assign
         the nth element to zones to z.
      a. If the d_d field is down(n) assign
         the nth element to downs to d.
      a. If the form field is formation(n) assign
         the nth element to formations to fo.
      a. If the cover field is cover(n) assign
         the nth element to covers to c.
    H. Assign front_by_hash_ar[f,h] + 1 to
       front_by_hash_ar[f,h].
    I. Assign front_by_zone_ar[f,z] + 1 to
       front_by_zone_ar[f,z].
    J. Assign front_by_down_ar[f,d] + 1 to
       front_by_down_ar[f,d].
    K. Assign front_by_formation_ar[f,fo] + 1 to
       front_by_formation_ar[f,fo].
```

L. Assign cover_by_down_ar[c,d] + 1 to
   cover_by_down_ar[c,d].
M. Assign cover_by_hash_ar[c,h] + 1 to
   cover_by_hash_ar[c,h].
N. Assign cover_by_zone_ar[c,z] + 1 to
   cover_by_down_ar[c,z].
O. Assign cover_by_formation_ar[c,fo] + 1 to
   cover_by_formation_ar[c,fo].
P. Assign cover_by_front_ar[c,f] + 1 to
   cover_by_front_ar[c,f].
Q. Assign what the next_dp field points to
   to tempptr.

1.4.2 INIT_MATRIX_ARRAYS

This procedure initializes each element of the matrix value arrays to zero.

```
Local variables
f of type fronts
h of type hashmarks
z of type zones
d of type downs
fo of type formations
c of type covers
```

```
ALGORITHM
  1. For all the fronts do
    A. For all the hashmarks do
      a. Assign zero to the elements of
           front_by_hash_ar.

  2. For all the fronts do
    A. For all the zones do
      a. Assign zero to the elements of
           front_by_zone_ar.

  3. For all the fronts do
    A. For all the downs do
      a. Assign zero to the elements of
           front_by_down_ar.

  4. For all the fronts do
    A. For all the formations do
      a. Assign zero to the elements of
           front_by_formation_ar.

  5. For all the covers do
    A. For all the downs do
      a. Assign zero to the elements of
           cover_by_down_ar.

  6. For all the covers do
    A. For all the hashmarks do
      a. Assign zero to the elements of
           cover_by_hash_ar.
```

```
7. For all the covers do
  A. For all the zones do
    a. Assign zero to the elements of
cover_by_zone_ar.

8. For all the covers do
  A. For all the formations do
    a. Assign zero to the elements of
        cover_by_formation_ar.

9. For all the covers do
  A. For all the fronts do
    a. Assign zero to the elements of
        cover_by_front_ar.
```

1.4.3 <u>ASSIGN_NAMES</u>

This procedure puts the names that are to be printed in the first column of the matrix charts into the "names" arrays.

ALGORITHM

1. Assign the name of each front to an element of frontnames.

2. Assign the name of each formation to an element of formations.

3. Assign the name of each down to an element of downnames.

4. Assign the name of each zone to an element of zonenames.

5. Assign the name of each hash mark to an element of hashnames.

1.4.4 COLUMN_TOTALS

This procedure figures the totals for the columns in the matrix charts.

```
Local variables
  i of type integer
  h of type hashmarks
  f of type fronts
  c of type covers

ALGORITHM
  1. For i from 1 to 16 do
    A. Assign zero to front_totals[i].

  2. For i from 1 to 13 do
    A. Assign zero to cover_totals[i].

  3. Assign zero to i.

  4. For f from okie to otherfront do
    A. Assign i + 1 to 1.
    B. For h from right to other hash do
      a. Assign front_totals[i] + front_by_hash_ar[f,h]
         to front_totals[i].

  5. Assign zero to i.

  6. For c from one to othercover do
    A. Assign i + 1 to 1.
    B. For f from okie to otherform do
      a. Assign cover_totals[i] + cover_by_front_ar[c,f]
         to cover_totals[i].
```

1.4.5 COLUMN_PERCENTS

This procedure finds the percentage of times that each front and cover is used.

```
Local variables
  tct of type integer
  tft of type integer
  i of type integer

ALGORITHM
  1. Assign zero to tct and tft.

  2. For i from 1 to 17 do
    A. Assign tft plus front_totals[i] to tft.

  3. For i from 1 to 18 do
    A. Assign ((front_totals[i] / tft) * 100) rounded
       off to the closest integer to frontpercents[i].

  4. Assign tft to frontpercents[18].

  5. For i from 1 to 14 do
    A. Assign tct plus cover_totals[i] to tct.

  6. For i from 1 to 14 do
    A. Assign ((cover_totals[i] / tct) * 100) rounded
       off to the closest integer to coverpercents[i].

  7. Assign tct to coverpercents[15].
```

1.5 WHAT_DCHART

This procedure finds out if the user wants list charts of matrix charts.

```
Local variables
  typedcharts of type char.
  another_d of type char
  dfile of type char
  do_d_again of type boolean.

ALGORITHM:
  1. Repeat until all charts needed have been made
           (do_d_again is false)
    A. Assign false to do_d_again.
    B. Ask the user if he wants list charts or
        matrix charts.
       Use this menu;
        L       for LIST CHARTS
        M       for MATRIX CHARTS
    C. Assign the answer to typedcharts.
    D. If the answer is M then
      a. Ask if the charts are to be to the
         terminal or to a file.
      b. Assign the answer to dfile.
      c. If the chart is to be to a file then
        i. If dfile_set is false then
          ii. Rewrite the outfile to file_out4.
          iii. Assign dfile_set to true.
      d. If to a file then
        i. Call procedure matrix_dcharts
           passing outfile.
          else
        ii. Call procedure matrix_dcharts
            passing tty.
        else
      e. Call procedure dlist_chart.
    E. Ask the user if he wants another defensive
       chart.
      a. If he does then
        i. Assign true to do_d_again.
        else
        ii. Assign false to do_d_again.
```

2 REED_D_PLAYS

This procedure asks the user what information he wants in the program to build the charts that he wants. He can use only newly entered plays, only plays that are all ready on file, of a combination of both.

```
Local variables
  done of type boolean
  finished of type char.
  old_dplays of type char.
  more_d of type char.

ALGORITHM:
  1. Assign false to done.

  2. Ask the user if he wants to include a
     previously built file of plays.

  3. If he does then
    A. Call procedure reed_oldd_plays.

  4. Ask the user if he wants to add new
     plays to the program.

  5. If he does then
    A. Ask who the opponent is for this set of plays.
    B. Assign the answer to dot.

  6. Tell the user to enter the indicated information
     about  each new play that he wants to enter.

  7. Repeat until all the plays have been entered.
               (done = true)
    A. Ask the user for the following information
       about each play to be entered.
     a. Ask for "HASH MARK" assign it to dhash;
     b. Ask for "YARDAGE ZONE" assign it to dzone;
     c. Ask for "DOWN & DISTANCE" assign to dd_d;
     d. Ask for "OFFENSIVE FORMATION" assign to dform;
     e. Ask for "OFFENSIVE MOTION" assign to dmotion;
     f. Ask for "DEFENSIVE FRONT" assign to dfront;
     g. Ask for "VARIATION TO FRONT" assign to dvar;
     h. Ask for "SECONDARY COVERAGE" assign to dcover;
     i. Ask for "DEFENSIVE STUNT" assign to dstunt;
    j. Ask for "BLITZ" assign to dblitz;
  B. Call procedure build_d_list passing it dtemp_ptr;
  D. Ask the user if that was the last play to be entered
    a. If it is then
      i. Assign true to done.

8. Call procedure make_d_file.
```

3 REED_OLDD_PLAYS

This procedure is used to read any file of plays that has been built with a previous run of this program on the same team that is now being scouted.

```
 Local variables
      j of type integer
      ch of type char

ALGORITHM:
  1. Reset the infile to the name contained in file_ind.

  2. While you have not reached the end of the file do
    A. Read the information on the file into the proper
       variables and read off the spaces between the items.
    C. Call procedure build_d_list to put this information
       into the list of records of defensive plays.
```

4 DLIST_CHART

This procedure finds out what type play the user wants a chart for and calls the procedure to print all the plays that match it.

```
Local variables
  wantedptr of type d_ptr
  dkey of type integer
  dh_key of type character
  dz_key of type character
  ddd_key of type two_string
  dform_key of type eight_string
  dmotion_key of type three_string
  dfront_key of type eight_string
  dvar_key of type three_string
  dcover_key of type eight_string
  dstunt_key of type twelve_string
  dblitz_key of type twelve_string
  dfile of type character
  another_dl of type character

ALGORITHM
  1. Repeat until no more list charts are wanted
    A. Ask the user if he wants the files written to
       a file or to the terminal.
    B. If to a file then
      a. If the outfile has not been set then
        i. reset the outfile to a file named by the
           value of file_out4
        ii. Assign True to dfile_set
      b. Set write_to_dfile to true.
    C. Call procedure init_field.
    D. Assign 'N' to another_dl.
    E. Ask the user how many items he wants the
       chart keyed on and read the answer into
       numdkeys.
    F. While the answer is greater than 10 do
      a. Tell the user that there are only 10
         keys and ask how many he wants again.
        i. Assign the answer to numdkeys.
    G. Assign 1 to which_dkey.
    H. Create a new D_play_record with wantedptr
       pointing to it.
    I. Assign a flag character meaning not_selected
         tp each field of the new record.
    J. Assign nil to the next_dp field of the
       new record.
```

```
K. While there are more keys to be entered do
         (numdkeys is greater than 1)
  a. Call procedure init_field.
  b. If which_dkey = 1 then
    i. Assign "FIRST" to drequest.
     else
    i.i Assign "NEXT " to drequest.
  c. Assign zero to which_dkey.
  d. Ask the user "WHAT DO YOU WANT THE " drequest
     " KEY TO BE?".
  e. Write the following menu to the terminal.
     1: HASH MARK               5: FRONT
     2: DOWN and DISTANCE       6: COVERAGE
     3: YARDAGE ZONE            7: STUNTS
     4: OFFENSIVE FORMATION     8: BLITZES
     9: ALL PLAYS              10: MOTION
   f. Read the answer into dkey.
   g. If dkey = 1 then
     i. Assign "HASH MARK          " to field.
     ii. Ask the user which hash mark he wants.
     iii. Assign the answer to dh_key.
     iv. Assign dh_key to the hash field of the
         record that wantedptr points to.
   h. If dkey = 2 then
     i. Assign "DOWN and DISTANCE  " to field.
     ii. Ask the user which down he wants.
     iii. Assign the answer to ddd_key.
     iv. Assign ddd_key to the d_d field of the
         record that wantedptr points to.
   i. If dkey = 3 then
     i. Assign "YARDAGE ZONE       " to field.
     ii. Ask the user which yardage zone he wants.
     iii. Assign the answer to dz_key.
     iv. Assign dz_key to the zone field of the
         record that wantedptr points to.
   j. If dkey = 4 then
     i. Assign "OFFENSIVE FORMATION" to field.
     ii. Ask the user which formation he wants.
     iii. Assign the answer to dform_key.
     iv. Assign dform_key to the form field of the
         record that wantedptr points to.
   k. If dkey = 5 then
     i. Assign "FRONT              " to field.
     ii. Ask the user which front he wants.
     iii. Assign the answer to dfront_key.
     iv. Assign dfront_key to the front field of the
         record that wantedptr points to.
   l. If dkey = 6 then
```

```
    i. Assign "COVERAGE           " to field.
    ii. Ask the user which coverage he wants.
    iii. Assign the answer to dcover_key.
    iv. Assign dcover_key to the cover field of the
        record that wantedptr points to.
  m. If dkey = 7 then
    i. Assign "STRUT              " to field.
    ii. Ask the user which stunt he wants.
    iii. Assign the answer to dstunt_key.
    iv. Assign dstunt_key to the stunt field of the
        record that wantedptr points to.
  n. If dkey = 8 then
    i. Assign "BLITZ              " to field.
    ii. Ask the user which blitz he wants.
    iii. Assign the answer to dblitz_key.
    iv. Assign dblitz_key to the blitz field of the
        record that wantedptr points to.
  o. If dkey = 9 then
    i. Assign "ALL PLAYS          " to field.
  p. If dkey = 10 then
    i. Assign "MOTION             " to field.
    ii. Ask the user which motion he wants.
    iii. Assign the answer to dmotion_key.
    iv. Assign dmotion_key to the motion field of the
        record that wantedptr points to.
  q. Assign numdkeys - 1 to numdkeys.
L. If to a file then
  A. Call procedure print_list_chart passing
     wantedptr and outfile.
   else
  B. Call procedure print_list_chart passing
     wantedptr and tty.
M. Assign false to d_headed.
N. Ask the user if he wants another list chart.
  a. Assign the answer to another_dl.
```

5 PRINT_LIST_CHART(WP : D_PTR;var out_dv : text)

This procedure writes the plays that are in the linked list of plays that match the play wanted by the user.

Local variables
  tp of type d_ptr

```
ALGORITHM
  1. Assign d_anchor to tp.

  2. Call procedure d_heading
     passing out_dv.

  3. While tp is not nil do
    A. If the ot field of the wanted
       play is not-selected or the
       same as the ot field of the
       play that tp points to then
      a. If the hash field of the wanted
         play is not-selected or the
         same as the hash field of the
         play that tp points to then
        i. If the zone field of the wanted
           play is not-selected or the
           same as the zone field of the
           play that tp points to then
          i.a If the zone field of the wanted
              play is not-selected or the
              same as the zone field of the
              play that tp points to then
            i.b If the d_d field of the wanted
                play is not-selected or the
                same as the d_d field of the
                play that tp points to then
              i.c If the form field of the wanted
                  play is not-selected or the
                  same as the form field of the
                  play that tp points to then
                i.d If the motion field of the wanted
                    play is not-selected or the
                    same as the motion field of the
                    play that tp points to then
                  i.e If the front field of the wanted
                      play is not-selected or the
                      same as the front field of the
                      play that tp points to then
                i.f If the vars field of the wanted
                   play is not-selected or the
                   same as the vars field of the
                   play that tp points to then
                  i.g If the cover field of the wanted
                     play is not-selected or the
                     same as the cover field of the
                     play that tp points to then
                    i.h If the stunt field of the wanted
                       play is not-selected or the
                       same as the stunt field of the
                       play that tp points to then
                      i.i If the blitz field of the wanted
                         play is not-selected or the
                         same as the blitz field of the
                         play that tp points to then
                        i.k Write all the fields of the
                            play that tp points to to the
                            output.
                        i.l Call procedure draw_line
                            73 and out_dv.
B. Assign what the next_dp field points to to tp.
```

6 D_HEADING(var out_dv : text)

This procedure will print the proper heading on the top af any defensive chart that is requested to the outfile. It is called by print_d_frecord.

ALGORITHM
```
  1. Write the following heading to the output.
  |   |h|z|dd|        |   |        |   |        |         |         |
  |   |a|o|oi|  off   | m |        |   |        |         |         |
  |opp|s|n|ws|        | o |front   |var|cover   | stunt   |  blitz  |
  |   |h|e|nt|  form  |   |        |   |        |         |         |
```

7 PRINT_FRONTBYHASH_AR(var out_dv : text)

This procedure writes the front by hash mark matrix chart.

```
Local variables
  f of type fronts
  h of type hashmarks
  i of type integer

ALGORITHM
  1. Write "       FRONT BY HASH" to the output.

  2. Call procedure print_front_heading.
     passing out_dv.

  3. Assign zero to i.

  4. For h from right to otherhash do
    A. Assign i + 1 to i.
    B. Write hashnames[i] to the output.
    C. For f from okie to otherfront do
      a. Write front_by_hash_ar[f,h] to the output.
    D. Call procedure draw_line passing 74
       and out_dv.

  5. Call procedure print_front_totals
     passing out_dv.
```

7.1 MATRIX_DCHARTS(var out_dv : text)

This procedure asks the user which matrix charts he wants and calls the procedures to write the wanted charts.

```
Local variables
  tydmat of type integer.
  another_dm of type char.
  done of type boolean.

ALGORITHM
  1. Call procedure assign_names.

  2. Call procedure init_matrix_arrays.

  3. Call procedure set_matrix_array_vars.

  4. Call procedure column_totals.

  5. Call procedure column_percents.

  6. Repeat until finished with matrix charts
   A. Assign true to done.
   B. Write "ENTER THE NUMBER THAT INDICATES
      THE TYPE OF MATRIX CHART YOU WANT" to the
      terminal.
   C. Write the following menu to the terminal:
        1 for COVER BY FRONT
        2 for COVER BY FORMATION
        3 for COVER BY ZONE
        4 for COVER BY HASH MARK
        5 for COVER BY DOWN AND DISTANCE
        6 for FRONT BY FORMATION
        7 for FRONT BY DOWN AND DISTANCE
        8 for FRONT BY ZONE
        9 for FRONT BY HASH
       10 for ALL THE ABOVE CHARTS
   D. Read the answer into tydmat.
     a. If tydmat is 1 then
       i. Call procedure print_coverbyfront_ar.
          passing out_dv.
     b. If tydmat is 2 then
       i. Call procedure print_coverbyformation_ar.
          passing out_dv.
     c. If tydmat is 3 then
       i. Call procedure print_coverbyzone_ar.
          passing out_dv.
  d. If tydmat is 4 then
    i. Call procedure print_coverbyhash_ar.
       passing out_dv.
  e. If tydmat is 5 then
    i. Call procedure print_coverbydown_ar.
       passing out_dv.
  f. If tydmat is 6 then
    i. Call procedure print_frontbyformation_ar.
       passing out_dv.
  g. If tydmat is 7 then
    i. Call procedure print_frontbydown_ar.
       passing out_dv.
  h. If tydmat is 8 then
    i. Call procedure print_frontbyzone_ar.
       passing out_dv.
  i. If tydmat is 9 then
    i. Call procedure print_frontbyhash_ar.
       passing out_dv.
  j. If tydmat is 10 then
    i. Call procedure print_coverbyfront_ar.
       passing out_dv.
    ii. Call procedure print_coverbyformation_ar.
        passing out_dv.
    iii. Call procedure print_coverbyzone_ar.
         passing out_dv.
    iv. Call procedure print_coverbyhash_ar.
        passing out_dv.
    v. Call procedure print_coverbydown_ar.
       passing out_dv.
    vi. Call procedure print_frontbyformation_ar.
        passing out_dv.
    vii Call procedure print_frontbydown_ar.
        passing out_dv.
    viii Call procedure print_frontbyzone_ar.
         passing out_dv.
    ix. Call procedure print_frontbyhash_ar.
        passing out_dv.
E. Ask the user if he wants another matrix chart.
   If he does repeat this procedure.
```

7.2 PRINT_COVER_HEADING(var out_dv : text)

This procedure writes the heading to the cover by matrix charts.

```
ALGORITHM
  1. Write the following heading on one line:
    "    | 1 | 2 | 3 | 5 | 6 | M |MF |M2D|WMM|WM2|
     WM3|COM|ROB|other" to the output.

  2. Call procedure draw_line passing it 62
     and out_dv.
```

7.3 PRINT_FRONT_HEADING

This procedure writes the heading to the front by matrix charts.

```
ALGORITHM
  1. If the chart is to be written to a file then
    A. Write the following heading on one line:
      "    |OKI|GRZ|SLS|SWS|SLT| 43|P43|C43| 42|42S|
       43S|44S|R62|EST|EWK|other" to the outfile.
   else
    A. Write the following heading on one line:
      "    |OKI|GRZ|SLS|SWS|SLT| 43|P43|C43| 42|42S|
       43S|44S|R62|EST|EWK|other" to the terminal.
  2. Call procedure draw_line passing it 74.
```

7.4 PRINT_FRONT_TOTALS(var out_dv : text)

This procedure is the one that writes the totals at the bottom of the front matrix charts.

Local variable
  i of type integer

```
ALGORITHM
  1. Write four spaces to the output.

  2. For i from 1 to 17 do
    A. Write front_totals[i] to the output.

  3. Call procedure print_front_column_percents
     passing it out_dv.
```

7.5 PRINT_COVER_TOTALS(var out_dv : text)

This procedure is the one that writes the totals at the bottom of the cover matrix charts.

```
Local variable
  i of type integer

ALGORITHM
  1. Write four spaces to the output.

  2. For i from 1 to 14 do
    A. Write cover_totals[i] to the output.

  3. Call procedure print_cover_column_percents
     passing out_dv.
```

7.6 PRINT_FRONT_COLUMN_PERCENTS(var out_dv :text)

This procedure writes the percentages to the front by matrix charts.

```
Local variables
  i of type integer

ALGORITHM
  1. Write four spaces to the output.

  2. For i from 1 to 17 do
    A. Write frontpercents[i] three spaces wide
       followed by a "%" sign to the output.

  3. Write "THE TOTAL NUMBER OF PLAYS ON THIS
     CHART IS " frontpercents[18] to the output.
```

7.7 PRINT_COVER_COLUMN_PERCENTS(var out_dv:text)

This procedure writes the percents on the bottom of the cover matrix charts.

```
Local variables
  i of type integer

ALGORITHM
  1. Write four spaces to the output.

  2. For i from 1 to 14 do
    A. Write coverpercents[i] three spaces wide
       followed by a "%" sign to the output.
    B. Write "THE TOTAL NUMBER OF PLAYS ON THIS
       CHART IS " coverpercents[15] to the output.
```

8 PRINT_FRONTBYZONE_AR(var out_dv:text)

This procedure writes the front by zone matrix chart.

```
Local variables
  f of type fronts
  z of type hashmarks
  i of type integer

ALGORITHM
  1. Write "       FRONT BY ZONE" to the output.

  2. Call procedure print_front_heading
     passing out_dv.

  3. Assign zero to i.

  4. For z from zone to otherzone do
    A. Assign i + 1 to i.
    B. Write zonenames[i] to the output.
    C. For f from okie to otherfront do
      a. Write front_by_zone_ar[f,z] to the output.
    D. Call procedure draw_line passing 74
       and out_dv.

  5. Call procedure print_front_totals
     passing out_dv.
```

9 PRINT_FRONTBYDOWN_AR(var out_dv : text)

This procedure writes the front by down matrix chart.

```
Local variables
  f of type fronts
  d of type downs
  i of type integer

ALGORITHM
  1. Write "       FRONT BY DOWN" to the output.

  2. Call procedure print_front_heading.
     passing out_dv.

  3. Assign zero to i.

  4. For d from ol to otherdown do
    A. Assign i + 1 to i.
    B. Write downname[i] to the output.
    C. For f from okie to otherfront do
      a. Write front_by_down_ar[f,d] to the output.
    D. Call procedure draw_line passing 74
       and out_dv.

  5. Call procedure print_front_totals
     passing out_dv.
```

10 PRINT_FRONTBYFORMATION_AR(var out_dv : text)

This procedure writes the front by hash mark matrix chart.

```
Local variables
  f of type fronts
  fo of type formations
  i of type integer

ALGORITHM
  1. Write "       FRONT BY FORMATION" to the output.

  2. Call procedure print_front_heading
     passing out_dv.

  3. Assign zero to i.

  4. For fo from spla to otherform do
    A. Assign i + 1 to i.
    B. Write formationnames[i] to the output.
    C. For f from okie to otherfront do
      a. Write front_by_formation_ar[f,fo] to the output.
    D. Call procedure draw_line passing 74
       and out_dv.

  5. Call procedure print_front_totals
     passing out_dv.
```

11 PRINT_COVERBYDOWN_AR(var out_dv:text)

This procedure writes the cover by down matrix chart.

```
Local variables
  c of type covers
  d of type downs
  i of type integer

ALGORITHM
  1. Write "      COVER BY DOWN" to the output.

  2. Call procedure print_cover_heading
     passing out_dv

  3. Assign zero to i.

  4. For d from ol to otherdown do
    A. Assign i + 1 to i.
    B. Write downnames[i] to the output.
    C. For c from one to othercover do
      a. Write cover_by_down_ar[c,d] to the output.
    D. Call procedure draw_line passing 62
       and out_dv.

 5. Call procedure print_cover_totals
    passing out_dv.
```

12 PRINT_COVERBYHASH_AR(var out_dv:text)

This procedure writes the cover by hash matrix chart.

```
Local variables
  c of type covers
  h of type hashmarks
  i of type integer

ALGORITHM
  1. Write "      COVER BY HASH" to the output.

  2. Call procedure print_cover_heading
     passing out_dv.

  3. Assign zero to i.

  4. For h from right to otherhash do
     A. Assign i + 1 to i.
     B. Write hashnames[i] to the output.
     C. for c from one to othercover do
       a. Write cover_by_hash_ar[c,h] to the output.
     D. Call procedure draw_line passing 62
        and out_dv.

 5. Call procedure print_cover_totals
    passing out_dv.
```

13 PRINT_COVERBYZONE_AR(var out_dv:text)

This procedure writes the cover by zone matrix chart.

```
Local variables
  c of type covers
  z of type zones
  i of type integer

ALGORITHM
  1. Write "      COVER BY ZONE" to the output.

  2. Call procedure print_cover_heading
     passing out_dv.

  3. Assign zero to i.

  4. For z from zone to otherzone do
    A. Assign i + 1 to i.
    b. Write zonenames[i] to the output.
    C. for c from one to othercover do
      a. Write cover_by_zone_ar[c,z] to the output.
   D. Call procedure draw_line passing 62
      and out_dv.

 5. Call procedure print_cover_totals
    passing out_dv.
```

14 PRINT_COVERBYFRONT_AR(var out_dv:text)

This procedure writes the cover by front matrix chart.

```
Local variables
  c of type covers
  f of type fronts
  i of type integer

ALGORITHM
  1. Write "      COVER BY FRONT" to the output.

  2. Call procedure print_cover_heading
     passing out_dv.

  3. Assign zero to i.

  4. For f from okie to otherfront do
    A. Assign i + 1 to i.
    B. Write frontnames[i] to the output.
    C. for c from one to othercover do
      a. Write cover_by_front_ar[c,f] to the output.
    D. Call procedure draw_line passing 62
       and out_dv.

 5. Call procedure print_cover_totals
    passing out_dv.
```

15 PRINT_COVERBYFORMATION_AR(var out_dv:text)

This procedure writes the cover by formation matrix chart.

```
Local variables
  c of type covers
  f of type formations
  i of type integer

ALGORITHM
  1. Write "      COVER BY FORMATION" to the output.

  2. Call procedure print_cover_heading
     passing out_dv.

  3. Assign zero to i.

  4. For f from spla to otherform do
    A. Assign i + 1 to i.
    B. Write formationnames[i] to the output.
    C. for c from one to othercover do
      a. Write cover_by_formation_ar[c,f] to the output.
    D. Call procedure draw_line passing 62
       and out_dv.

 5. Call procedure print_cover_totals
    passing out_dv.
```

16 <u>BUILD_D_LIST(VAR TEMP_DPTR:D_PTR)</u>

This procedure is used to assign the proper values to the fields in the record of defensive plays and to add the new record to the linked list of plays. Each new record is added to the end of the linked list so that the charts when written will have the plays in the same order that they were entered.

ALGORITHM:

```
1. Increment total_dplays.

2. Create a pointer to a new defensive play record.

3. If this is the first play then
  A. Assign it to the anchor.
  B. Assign it to temp_dptr.
 else
  C. Assign it to the next play field of the record
     that temp_dptr is pointing to.
  D. Assign it to temp_dptr.

4. Assign the defensive variables to the proper
   field of the new record.

5. Assign nil to the next record field of the
   new record.
```

18 <u>DRAW_LINE(LENGTH : INTEGER;VAR OUT_DV:TEXT)</u>

This procedure draws lines on the output charts.

```
Local variable
  i of type integer

ALGORITHM
  1. For i assigned 1 to length
    A. Print length dashes to the output device.
```

19 MAKE_D_FILE

This procedure writes the defensive records to a file so they can be used in a later run of the program.

Local variable
 df_ptr of type d_ptr

```
ALGORITHM:
  1. Rewrite the outfile to the name
     contained in file_out2.

  2. Assign d_anchor to df_ptr.
     Repeat until all the plays have been written.
          ( df_ptr = nil.)
    A. Write to the outfile the values of the fields
       of the record that df_ptr points to in the
       following order.  They are to be on the same
       line and each followed by a space.
       1. ot
       2. zone
       3. d_d
       4. form
       5. motion
       6. front
       7. vars
       8. cover
       9. stunt
      10. blitz
      11. hash
    B. Assign what the next_dp field of the record
       points to df_ptr.
```

20 <u>INIT_FIELD</u>

This procedure initializes the elements of the array that holds the names of the charts to spaces.

Local variable
  I of type integer.

```
ALGORITHM
  1. For i from 1 to 20
    A. Assign a space to field indexed by i.
```

CHART O
O-SCOUT

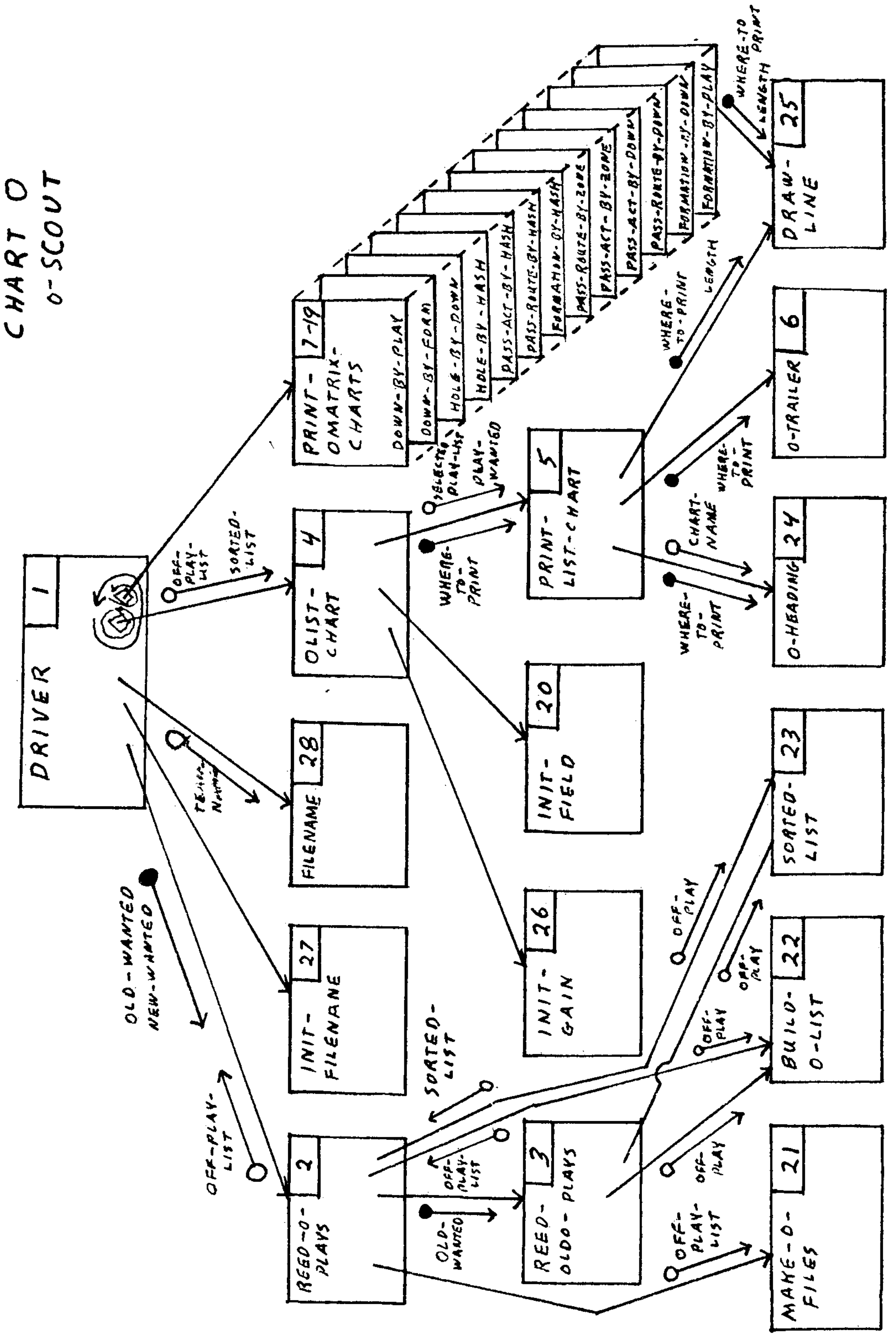


CHART 1
DRIVER

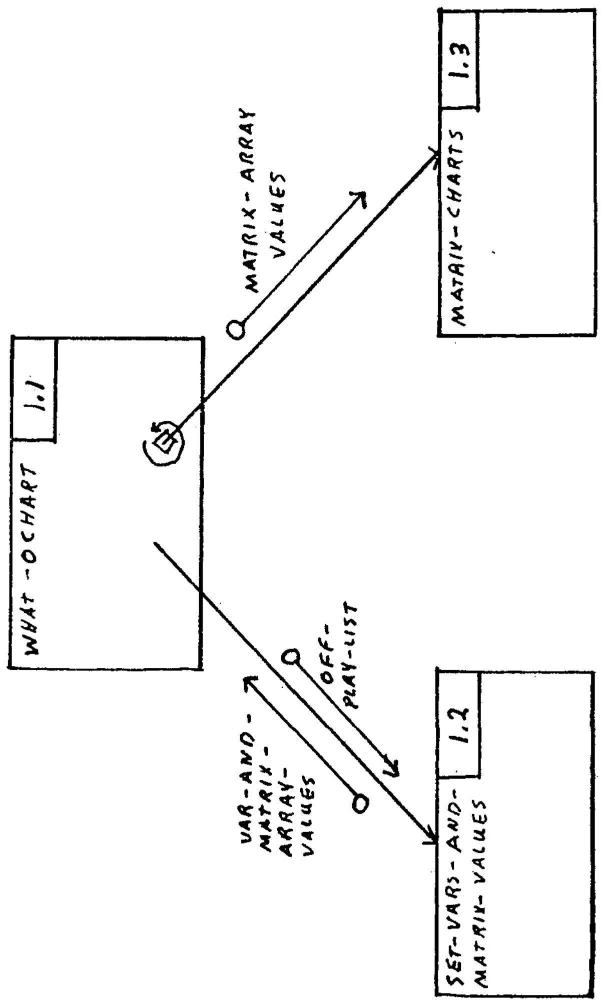


CHART 1.2

SET-VARS-AND-OMATRIX-VALUES

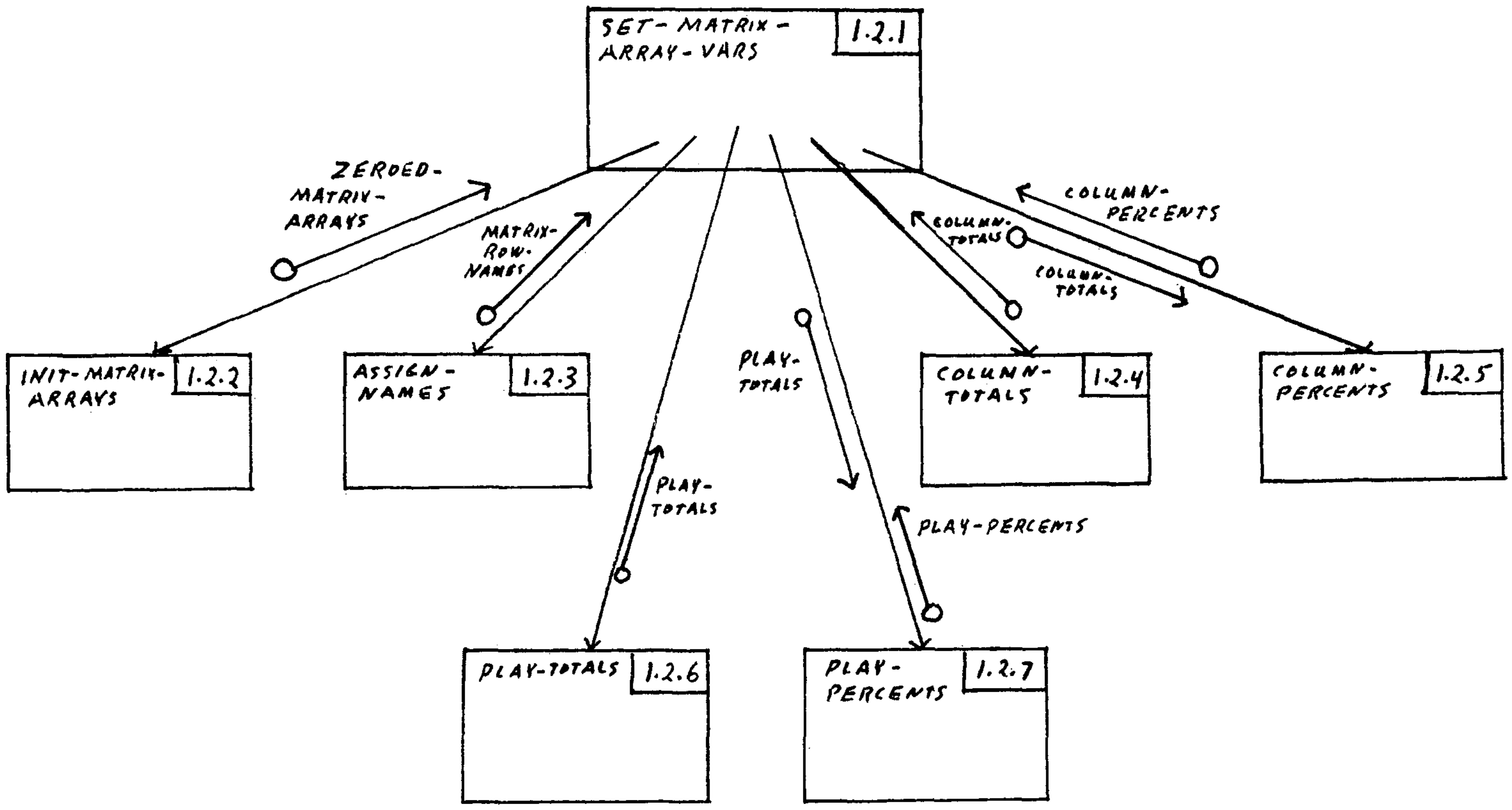


CHART 6
OFF-LIST-CHART-TRAILER

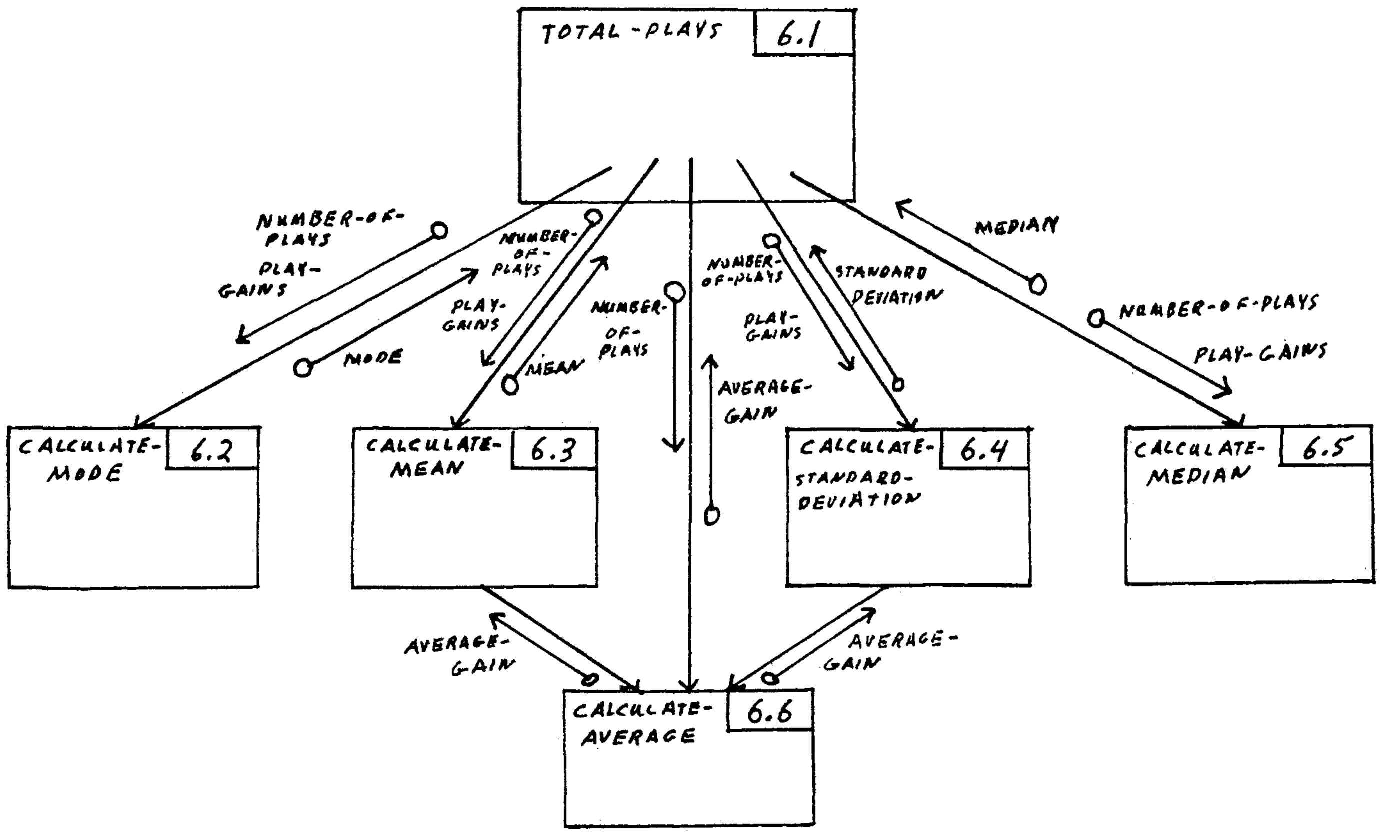
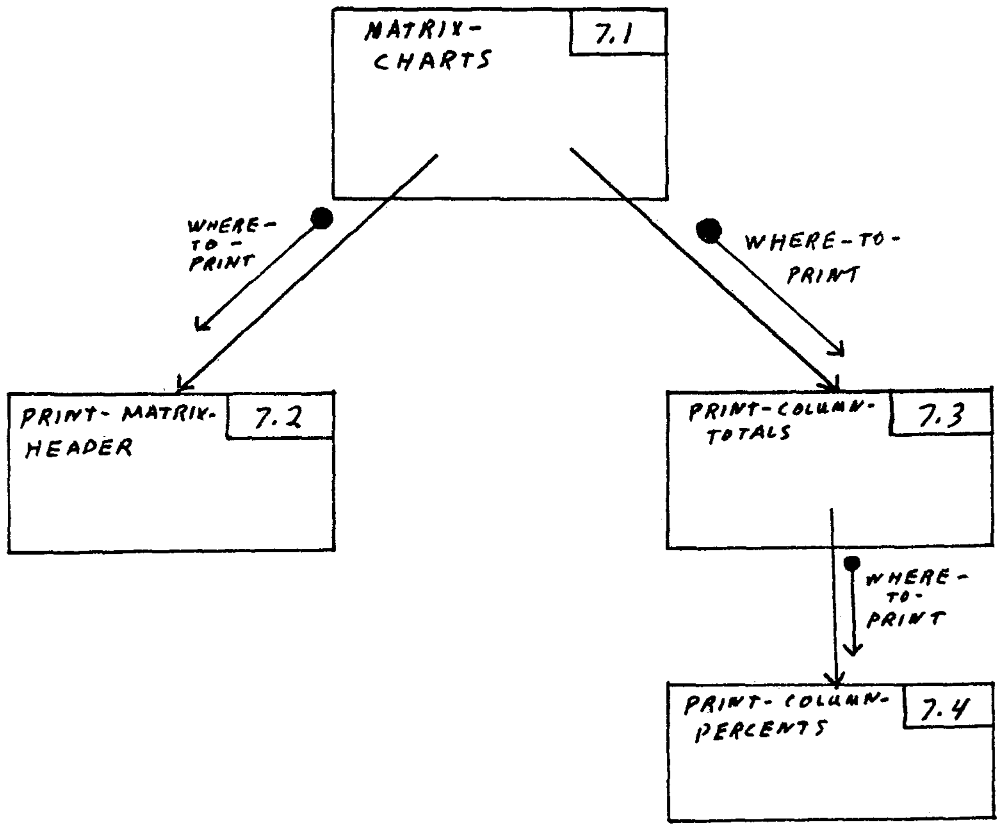


CHART 7-19

PRINT-OMATRIX-CHARTS



o-SCOUT
CONSTANTS

```
   play1 = '59      '      play17 = '53      '
   play2 = '59S     '      play18 = '52      '
   play3 = '58      '      play19 = '41T     '
   play4 = '58S     '      play20 = '40T     '
   play5 = '57      '      play21 = '39      '
   play6 = '56      '      play22 = '39S     '
   play7 = '55      '      play23 = '38      '
   play8 = '55B     '      play24 = '38S     '
   play9 = '55C     '      play25 = '34      '
   play10 = '55R     '     play26 = '35      '
   play11 = '55CR    '     play27 = '31      '
   play12 = '54      '     play28 = '30      '
   play13 = '54B     '     play29 = '15      '
   play14 = '54C     '     play30 = '14      '
   play15 = '54R     '     play31 = '13      '
   play16 = '54CR    '     play32 = '12      '
```

These are the offensive plays that the Grizzly coaching staff wants included. To change the plays that are handled by the program all that is necessary is to change these to the proper plays and change the heading on the output charts.

```
   form1 = 'PRI     '      form9 = 'SRS     '
   form2 = 'PLI     '      form10 = 'SLS     '
   form3 = 'PRQ     '      form11 = 'SPRA    '
   form4 = 'PLQ     '      form12 = 'SPLA    '
   form5 = 'SRI     '      form13 = 'SPRB    '
   form6 = 'SLI     '      form14 = 'SPLB    '
   form7 = 'SRQ     '      form15 = 'TWRI    '
   form8 = 'SLQ     '      form16 = 'TWLI    '
```

These are the formations used by the Grizzlies.

```
   passact1 = '500     '  route1 = '1      '
   passact2 = '700     '  route2 = '2      '
   passact3 = '900     '  route3 = '3      '
   passact4 = '80      '  route4 = '3SW    '
   passact5 = '90      '  route5 = '4      '
   passact6 = 'BOOT    '  route6 = '4SW    '
   passact7 = 'SCREEN  '  route7 = '5      '
   passact8 = 'YSC     '  route8 = '5SW    '
```
These are the pass actions and pass routes used by the Grizzlies.

TYPES

O_PTR a pointer to a record of offensive plays
TEAM_NAME a packed array of 15 elements to be
used to name files.
TWO_STRING a packed array of two elements
THREE_STRING a packed array of three elements
FOUR_STRING a packed array of four elements
FIVE_STRING a packed array of five elements
EIGHT_STRING a packed array of eight elements

O_PLAY_RECORD the record for an offensive play
containing the following fields:

- OT of type three_string to hold the opposing team
- HASH of type char to hold the hash mark
- ZONE of type char to hold the yardage zone
- D_D of type two_string to hold the down and distance
- FORM of type eight_string to hold the formation
- R_P of type char to hold run or pass
- PLAY of type eight_string to hold the play
- POS_BC of type two_string to hold the ball carriers position
- B_C of type two_string to hold the ball carriers number
- HOLE of type char to hold the hole number
- S_W of type char to hold strong or weak side
- RESULT of type integer to hold the gain of loss
- P_ACT of type eight_string to hold the pass action
- P_RESLT of type three_string to hold the pass result
- REC of type two_string to hold the receivers number
- P_PATTERN of type eight_string to hold the pass pattern
- P_DEPTH of type two_string to hold the depth the pass is thrown
- P_ZONE of type char to hold the zone to which the pass is thrown
- REC_P of char to hold the position of the receiver
- DRIVE_NO of type integer to hold the number of the drive the play was in
- PLAY_NO of type integer to hold the number of the play in the drive
- NEXT_OP of type o_ptr to point to the next play

```
HOLES = (hnine,hseven,hfive,hthree,hone,hzero,htwo,
         hfour,hsix,height,pass,otherhole)
DOWNS = (ol,om,os,og,sl,sm,ss,sg,tl,tm,ts,tg,
         fl,fm,fs,fg,otherdown)
HASHMARKS = (right,middle,left,otherhash)
FORMATIONS = (pri,pli,prq,plq,sri,sli,srq,slq,
              srs,sls,spra,spla,sprb,splb,twri,
              twli,otherform)
PLAYS = (p59,p59s,p58,p58s,p57,p56,p55,p55b,p55c,
         p55r,p55cr,p54,p54b,p54c,p54r,p54cr,p53,
         p52,p41t,p40t,p39,p39s,p38,p38s,p34,p35,
         p31,p30,p15,p14,p13,p12,otherplay)
PASS_ACTIONS = (a500,a700,a900,a80,a90,abot,ascr,
                aysc,otherpassaction)
ROUTES = (r1,r2,r3,r3sw,r4,r4sw,r5,r5sw,otherroute)
ZONES = (zone,ztwo,zthree,zfour,zfive,zsix,otherzone)
```

GLOBAL VARIABLES

DOWN_BY_PLAY_AR an array of downs by plays
  of type integer
TDOWNCHART an array of 17 integers
PDOWNCHART an array of 18 integers
HOLE_BY_FORMATION_AR an array of holes by
  formations of type integer
THOLECHART an array of 12 integers
PHOLECHART an array of 13 integers
HOLE_BY_DOWN_AR an array of holes by downs
  of type integer
HOLE_BY_HASH_AR an array of holes by
  hashmarks of type integer
PACT_BY_HASH_AR an array of pass_actions
  by hashmarks of type integers
TPACTCHART an array of 9 integers
PPACTCHART an array of 10 integers
ROUTE_BY_HASH_AR an array of routes by
  hashmarks of type integer
TROUTECHART an array of 9 integers
PROUTECHART an array of 10 integers
FORM_BY_HASH_AR an array of formations
  by hashmarks of type integer
TFORMCHART an array of 17 integers
PFORMCHART an array of 18 integers
ROUTE_BY_ZONE_AR an array of routes by
  zones of type integer
PACT_BY_ZONE_AR an array of pass_actions
  by zones of type integer
PACT_BY_DOWN_AR an array of pass_actions
  by downs of type integer
ROUTE_BY_DOWN_AR an array of routes by
  downs of type integer
FORM_BY_DOWN_AR an array of formations
  by downs of type integer
FORM_BY_PLAY_AR an array of formations
  by plays of type integer
TPLAY an array of 33 integers
PPLAY an array of 33 integers
DOWNNMAES an array of 17 four_string
FORMATIONNAMES an array of 17 four_string
HASHNAMES an array of 4 four_string
ZONENAMES an array of 7 four_string

```
PLAYNAMES an array of 33 four_string
INFILE,OUTFILE of type text
O_ANCHOR of type o_ptr to point to the first
   offensive record
O_RECORD of type o_play_record, the record of an
   offensive play
FILE_INO,FILE_IND,FILE_OUT1,FILE_OUT2 of type team_name
   to hold the names of the input and output files
ALLDONE of type char
OCURRENT,T_PTR of type o_ptr
T_PTR of type o_ptr
SO_CURRENT,SO_ANCHOR of type o_ptr
SORTED of type boolean
OOT of type three_string to hold the opposing team
OHASH of type char to hold the hash mark
OZONE to hold the yardage zone
OD_D of type two_string to hold the down and
  distance
OFORM of type eight_string to hold the
  formation of the play
OR_P of type char to hold whether the play
  is a run or a pass
OPLAY of type eight_string to hold the play
OPB_C of type two_string to hold the position
  of the ball carrier
OB_C of type two_string to hold the number of
  the ball carrier
OHOLE of type char to hold the hole the play
  is run to
OS_W of type char to hold weak or strong side
ORESULT of type integer to hold the loss or
  gain of the play
OP_ACT of type eight_string to hold the action
  of the quarterback on a pass play
OP_RESLT of type three_string to hold the result
  of a pass play
OREC of type two_string to hold the receivers
  number
OP_ZONE of type char to hold the zone that a
  pass is thrown to.
OREC_P      of type char to hold the position of the
  receiver
OP_PATTERN of type eight_string to hold the pattern
  run by the receiver
OP_DEPTH of type two_string to hold the distance
  the pass was thrown
ODRIVE_NO of type integer to hold the drive number
   of the play
```

OPLAY_NO of type integer to hold which play of
   the drive
OC of type char to hold if charts are wanted or not
TOTAL_OPLAYS of type integer to hold the total
   number of plays entered
OLD_OPLAYS of type char to hold if a file of old
   plays should be read or not
O_HEADED of type boolean to hold if the charts have
   headers or not
WRITE_TO_OFILE of type boolean to hold if the charts
   are to be written to a file or to the ternimal
FIELD a packed array of 20 characters to hold the
   name of the chart being produced
TOTAL of type integer holds the total yards for
   the chart being produced
NUMPLAYS of type integer to hold the number
  of plays on the chart being produced
TOTAL_OPLAYS of type integer
GAIN an array -100 to 100 of 201 elements
ODD_KEY of type boolean
NUMKEYS of type integer
MANY_KEYS of type boolean
FIRST of type boolean
REQUEST of type five_string
WHICH_KEY of type integer
TPL_PTR of type o_ptr
FILE_SET of type boolean
OMMENU, OLMENU of type boolean

PROCEDURES

1 MAIN

This is the main driver for the rest of the program.

ALGORITHM
```
 1. Assign false to ommenu.

 2. Assign false to olmenu.

 3. Assign false to file_set.

 4. Print the following heading "o-SCOUT"

      to the terminal.

 5. Assign nil to o_anchor, so_anchor,t_ptr.

 6. Assign nil to t1_oanchor,t12_oanchor.

 7. Assign false to o_headed.

 8. Call procedure init_filename.

 9. Reset the input to the terminal.

10. Ask the user what team is being scouted.

11. Assign his answer to file_out1.

12. Assign file_out1 to file_ino.

13. Call procedure filename.

14. Call procedure reed_o_plays.

15. Ask the user if he wants offensive charts.
     A. If he does then
       a. Call procedure what_ochart.

17. Ask the user if he wants to run the
    program again.

18. Assign the answer to alldone.
```

1.1 WHAT_OCHART

This procedure finds out if the user wants matrix charts of list charts, and if the are to to be written to a file or to the termianl.

```
Local variables
  another_o, wfile, typecharts,
  more_o of type char
  do_o_again of type boolean

ALGORITHM
  1. Repeat until no more charts are wanted.
    A. Call procedure init_field.
    B. Call procedure init_gain.
    C. Ask if list or matrix charts are wanted.
    D. Assign his answer to typecharts.
    E. If martix charts wanted then
      a. Ask if charts are to be to a file or to the terminal.
      b. Assign answer to wfile.
      c. If they are to a file then
        i. If file_set is false then
          ii. Rewrite the outfile to file_out3.
          iii. Assign true to file_set.
        iv. Assign true to write_to_file.
      d. If they are to a file then
        i. Call procedure matrix_charts passing outfile.
        else
        ii. Call procedure matrix_charts passing tty.
      else
      e. Call procedure olist_chart.
    F. Ask if more offensive charts are wanted.
```

1.2.1 SET_MATRIX_ARRAY_VARS

This procedure puts the proper values into the elements of the matrix arrays.

```
Local variables
  tempptr of type o_ptr
  d of type downs
  f of type formations
  hm of type hashmarks
  z of type zones
  pl of type plays
  pa of type pass_actions
  r of type routes
  ho of type holes

ALGORITHM
  1. Assign o_anchor to tempptr.

  2. While tempptr is not nil do
    A. Assign otherdown to d.
    B. Assign otherform to f.
    C. Assign otherhash to hm.
    D. Assign otherzone to z.
    E. Assign otherplay to pl.
    F. Assign otherpassaction to pa.
    G. Assign otherroute to r.
    H. Assign otherhole to ho.
    I. Check all the necessary fields of the
       play that tempptr points to
      a. If the d_d field is play(n) assign
         the nth element of plays to d.
      b. If the form field is form(n) assign
         the nth element of formations to f.
      c. If the hash field is hash(n) assign
         the nth element of hashmarks to hm.
      d. If the play field is play(n) assign
         the nth element of plays to pl.
      e. If the p_act field is passact(n) assign
         the nth element of pass_actions to pa.
      f. If the p_pattern field is route(n)
         assign the nth element of pass_actions
         to r.
      g. If the hole field is (n) assign
         the nth element of holes to ho.
      h. If the r_p field is "P" assign
         pass to ho.
```

```
i. If the zone field is zone(n) assign
   the nth element of zones to z.
j. Assign down_by_play_ar[d,pl] + 1 to
   down_by_play_ar[d,pl].
k. Assign form_by_hash_ar[f,hm] + 1 to
   form_by_hash_ar[f,hm].
l. Assign form_by_down_ar[f,d] + 1 to
   form_by_down_ar[f,d].
m. Assign form_by_play_ar[f,pl] + 1 to
   form_by_play_ar[f,pl].
n. Assign hole_by_formation_ar[ho,f] + 1 to
   hole_by_formation_ar[ho,f].
o. Assign hole_by_down_ar[ho,d] + 1 to
   hole_by_down_ar[ho,d].
p. Assign hole_by_hash_ar[ho,hm] + 1 to
   hole_by_hash_ar[ho,hm].
q. If the r_p field is "P" then
  i. Assign pact_by_hash_ar[pa,hm] + 1 to
     pact_by_hash_ar[pa,hm].
  ii. Assign route_by_hash_ar[r,hm] + 1 to
      route_by_hash_ar[r,hm].
  iii. Assign route_by_zone_ar[r,z] + 1 to
       route_by_zone_ar[r,z].
  iv. Assign pact_by_zone_ar[pa,z] + 1 to
      pact_by_zone_ar[pa,z].
  v. Assign pact_by_down_ar[pa,d] + 1 to
     pact_by_down_ar[pa,d].
  vi. Assign route_by_down_ar[r,d] + 1 to
      route_by_down_ar[r,d].
r. Assign the next_op field to tempptr.
```

1.2.2 INIT_MATRIX_ARRAYS

This procedure initializes the elemenats of the matrix arrays to zero.

```
Local variables
   d of type downs
   f of type formations
   hm of type hashmarks
   z of type zones
   pl of type plays
   pa of type pass_actions
   r of type routes
   ho of type holes

ALGORITHM
  1. For d from ol to otherdown do
    A. For pl from p59 to otherplay do
      a. Assign zero to down_by_play_ar[d,pl].

  2. For ho from hnine to otherhole do
    A. For f from pri to otherform do
      a. Assign zero to hole_by_form_ar[ho,f].

  3. For ho from hnine to otherhole do
    A. For d from ol to otherdown do
      a. Assign zero to hole_by_down_ar[ho,d].

  4. For ho from hnine to otherhole do
    A. For hm from right to otherhash do
      a. Assign zero to hole_by_hash_ar[ho,hm].

  5. For pa from a500 to otherpassaction do
    A. for hm from right to otherhash do
      a. Assign zero to pact_by_hash_ar[pa,hm].

  6. For r from r1 to otherroute do
    A. for hm from right to otherhash do
      a. Assign zero to route_by_hash_ar[r,hm].

  7. For f from pri to otherform do
    A. for hm from right to otherhash do
      a. Assign zero to form_by_hash_ar[f,hm].

  8. For r from zone to otherzone do
    A. for z from zone to otherzone do
      a. Assign zero to route_by_zone_ar[r,z].
```

```
 9. For pa from a500 to otherpassaction do
    A. for z from zonet to otherzone do
      a. Assign zero to pact_by_zone_ar[pa,z].

10. For pa from a500 to otherpassaction do
    A. for d from ol to otherdown do
      a. Assign zero to pact_by_down_ar[pa,d].

11. For r from r1 to otherroute do
    A. for d from ol to otherdown do
      a. Assign zero to route_by_down_ar[r,d].

12. For f from pri to otherform do
    A. for d from ol to otherdown do
      a. Assign zero to form_by_down_ar[f,d].

13. For f from pri to otherform do
    A. for pl from p59 to otherplay do
      a. Assign zero to form_by_play_ar[f,pl].
```

1.2.3 ASSIGN_NAMES

This procedure assigns the names to the "name" arrays so they can be printed in the first column of the matrix arrays. ALGORITHM

1. Assign the names of the downs to each element of dowmnames.

2. Assign the names of the formations to each element of formationnames.

3. Assign the names of the hash marks to each element of hashnames.

4. Assign the names of the zones to each element of zonenames.

5. Assign the names of the plays to each element of playnames.

1.2.4 COLUMN_TOTALS

This procedure finds the totals for each column in the matrix charts.

```
Local variables
  i of type integer
  d of type downs
  f of type formations
  hm of type hashmarks
  pl of type plays
  pa of type pass_actions
  r of type routes
  ho of type holes

ALGORITHM
  1. For i from 1 to 17 do
    A. Assign zero to tdownchart[i].
    B. Assign zero to tformchart[i].

  2. For i from 1 to 9 do
    A. Assign zero to tpactchart[i].
    B. Assign zero to troutechart[i].

  3. For i from 1 to 12 do
    A. Assign zero to tholechart[i].

  4. Assign zero to i.

  5. For d from ol to otherdown do
    A. Assign i + 1 to i.
    B. For pl from p59 to otherplay do
      a. Assign tdownchart[i] + down_by_play_ar[d,pl]
         to tdownchart[i].

  6. Assign zero to i.

  7. For ho from hnine to otherhole do
    A. Assign i + 1 to i.
    B. For hm from right to otherhash do
      a. Assign tholechart[i] + hole_by_hash_ar[ho,hm]
         to tholechart[i].

  8. Assign zero to i.

  9. For pa from a500 to otherplay do
    A. Assign i + 1 to i.
```

```
   B. For hm from right to otherhash do
     a. Assign tpactchart[i] + pact_by_hash_ar[pa,hm]
        to tpactchart[i].

10. Assign zero to i.

11. For r from r1 to otherroute do
   A. Assign i + 1 to i.
   B. For hm from right to otherhash do
     a. Assign troutechart[i] + route_by_hash_ar[r,hm]
        to troutechart[i].

12. Assign zero to i.

13. For f from pri to otherform do
   A. Assign i + 1 to i.
   B. For hm from right to otherhash do
     a. Assign tformchart[i] + form_by_hash_ar[f,hm]
        to tformchart[i].
```

1.2.5 COLUMN_PERCENTS

This procedure finds the percentage of times that the items in each column of the matrix charts is used and puts that amount into the percent arrays.

```
Local variables
  i of type integer
  tdt of type integer
  tht of type integer
  tpt of type integer
  trt of type integer
  tft of type integer

ALGORITHM
  1. Assign zero to each of the local
     variables.

  2. For i from 1 to 17 do
    A. Assign tdt + tdownchart[i] to tdt
    B. Assign tft + tformchart[i] to tft

  3. For i from 1 to 17 do
    A. Assign ((tdownchart[i] / tdt) * 100)
       rounded to the nearest integer to
       pdownchart[i].
    B. Assign ((tformchart[i] / tft) * 100)
       rounded to the nearest integer to
       pformchart[i].

  4. Assign tdt to pdownchart[18].

  5. Assign tft to pformchart[18].

  6. For i from 1 to 12 do
    A. Assign tht + tholechart[i] to tht.

  7. For i from 1 to 12 do
    A. Assign ((tholechart[i]) * 100)
       rounded to the nearest integer
       to pholechart[i].

  8. Assign tht to pholechart[13].

  9. For i from 1 to 9 do
    A. Assign tpt + tpactchart[i] to tpt.
```

```
   B. Assign trt + troutechart[i] to trt.

10. For i from 1 to 9 do
   A. Assign ((tpactchart / tpt) * 100)
      rounded off to the nearest integer
      to ppactchart[i].
   B. Assign ((troutechart /rt) * 100)0)
      rounded off to the nearest integer
      to proutechart[i].

11. Assign tpt to ppactchart[10].

12. Assign trt to proutechart[10].
```

1.2.6 PLAY_TOTALS

This procedure finds the total number of times that each play is run.

```
Local variables
   p of type plays
   d of type downs
   i of type integer

ALGORITHM
  1. For i from 1 to 33 do
    A. Assign zero to tplay[i].

  2. Assign zero to i.

  3. For p from p59 to otherplay do
    A. Assign i + 1 to i.
    B. For d from ol to otherdown do
      a. Assign tplay[i] + down_by_play_ar[i]
           to tplay[i].
```

1.2.7 PLAY_PERCENTS

This procedure finds the percentage of times that each play is used.

```
Local variables
  tplayt of type integer
  i of type integer

ALGORITHM
  1. Assign total_oplays to tplayt.

  2. For i from 1 to 33 do
    A. Assign ((tplay[i] / tplayt) * 100)
       rounded off to the closest integer
       to pplay[i].
```

1.3 MATRIX_CHARTS(var out_dv : text)

This procedure calls the procedures that set up the matrix array values and the ones to write the matrix charts that the used wants.

```
Local variables
    tymat of type integer to hold which type of
       matrix chart is wanted
    another_m of type char
    done of type boolean
    cform of type eight_string

ALGORITHM
  1. Call procedure assign_names.

  2. Call procedure init_matrix_arrays.

  3. Call procedure set_matrix_arrays.

  4. Call procedure colunm_totals.

  5. Call procedure column_percents.

  6. Call procedure play_totals.

  7. Call procedure play_percents.

  8. Repeat until done is true
    A. Assign true to done
    B. Write the following menu to the terminal:
       ENTER THE NUMBER THAT INDICATES THE TYPE OF
       MATRIX CHART THAT YOU WANT
          1 for HOLE BY DOWN CHART
          2 for HOLE BY HASH CHART
          3 for HOLE BY FORMATION CHART
          4 for PLAY BY DOWN CHART
          5 for PASS ACTION BY HASH CHART
          6 for PASS ROUTE BY HASH CHART
          7 for FORMATION BY HASH CHART
          8 for PASS ACTION BY ZONE CHART
          9 for PASS ROUTE BY ZONE CHART
         10 for PASS ACTION BY DOWN CHART
         11 for PASS ROUTE BY DOWN CHART
         12 for FORMATION BY DOWN CHART
         13 for FORMATION BY PLAY CHART
         14 for ALL OF THE CHARTS
   C. Read the answer into tymat
```

```
D. If tymat is 1 then
  a. Call procedure print_hole_by_down_ar
     passing out_dv.
E. If tymat is 2 then
  a. Call procedure print_hole_by_hash_ar
     passing out_dv.
F. If tymat is 3 then
  a. Call procedure print_hole_by_formation_ar
     passing out_dv.
G. If tymat is 4 then
  a. Call procedure print_down_by_play_ar
     passing out_dv.
H. If tymat is 5 then
  a. Call procedure print_pass_action_by_hash_ar
     passing out_dv.
I. If tymat is 6 then
  a. Call procedure print_pass_route_by_hash_ar
     passing out_dv.
J. If tymat is 7 then
  a. Call procedure print_formation_by_hash_ar
     passing out_dv.
K. If tymat is 8 then
  a. Call procedure print_pass_action_by_zone_ar
     passing out_dv.
L. If tymat is 9 then
  a. Call procedure print_pass_route_by_zone_ar.
     passing out_dv.
M. If tymat is 10 then
  a. Call procedure print_pass_action_by_down_ar
     passing out_dv.
N. If tymat is 11 then
  a. Call procedure print_pass_route_by_down_ar
     passing out_dv.
O. If tymat is 12 then
  a. Call procedure print_formation_by_down_ar
     passing out_dv.
P. If tymat is 13 then
  a. Call procedure print_formation_by_play_ar
     passing out_dv.
Q. If tymat is 14 then
  a. Call procedure print_hole_by_down_ar
     passing out_dv.
  b. Call procedure print_hole_by_hash_ar
     passing out_dv.
  c. Call procedure print_hole_by_formation_ar
     passing out_dv.
  d. Call procedure print_down_by_play_ar
     passing out_dv.
  e. Call procedure print_pass_action_by_hash_ar
     passing out_dv.
  f. Call procedure print_pass_route_by_hash_ar
     passing out_dv.
  g. Call procedure print_formation_by_hash_ar
     passing out_dv.
  h. Call procedure print_pass_action_by_zone_ar
     passing out_dv.
  i. Call procedure print_pass_route_by_zone_ar.
     passing out_dv.
  j. Call procedure print_pass_action_by_down_ar
     passing out_dv.
  k. Call procedure print_pass_route_by_down_ar
     passing out_dv.
  l. Call procedure print_formation_by_down_ar
     passing out_dv.
  m. Call procedure print_formation_by_play_ar
     passing out_dv.
R. Ask the user if he wants another matrix
   chart and read the answer into another_m.
S. If the answer is yes then set done to false.
```

2 REED_O_PLAYS

This procedure askes the user what information he wants in the program to build the charts that he wants. He can use only newly entered plays, only plays that are all ready on file, of a combination of both.

```
Local variables
  done of type boolean
  finished, more_o of type char

ALGORITHM:
  1. Assign false to done

  2. Ask the user if he wants to include a previously built
     file of plays.

  3. If the answer is yes then
    A. Call procedure reed_oldo_plays.

  4. Ask the user if he wants to add more plays.

  5. If the answer is yes then
    A. Ask for "THE OPPONENT" assign to oot.
    B. Tell the user to enter the indicated information
       for each play he wants to enter.
    C. Repeat until done equals true
      a. Ask for "DRIVE NUMBER" assign to odrive_no.
      b. Ask for "PLAY NUMBER" assign to oplay_no.
      c. Ask for "HASH MARK" assign to ohash.
      d. Ask for "YARDAGE ZONE" assign to ozone.
      e. Ask for "DOWN & DISTANCE" assign to od_d.
      f. Ask for "FORMATION" assign to oform.
      g. Ask for "RUN or PASS" assign to or_p.
      h. Ask for "PLAY" assign to oplay.
      i. If or_p is a run then
        i. Assign a "-" to op_act.
        ii. Assign a "-" to op_result.
        iii. Assign a "-" to orec_p.
        iv. Assign a "-" to orec.
        v. Assign a "-" to op_pattern.
        vi. Assign a "-" to op_zone.
        vii. Ask for "THE POSITION OF THE BALL CARRIER"
             assign to opb_c.
        viii. Ask for "BALL CARRIERS NUMBER" assign to ob_c.
      ix. Ask for "HOLE NUMBER" assign to ohole.
      x. Ask for "STRONG or WEAK SIDE" assign to os_w.
    j. If or_p is a pass then
      i. Assign a "-" to opb_c.
      ii. Assign a "-" to ob_c.
      iii. Assign a "-" to ohole.
      iv. Ask for "PASS ACTION" assign to op_act.
      v. Ask for "PASS RESULT" assign to pr_result.
      vi. Ask for "STRONG OR WEAK SIDE" assign to os_w.
      vii. Ask for "POSITION OF INTENDED RECIEVER"
           assign to orec_p.
      viii. Ask for "INDENDED RECEIVERS NUMBER"
            assign to orec.
      ix. Ask for "PATTERN RUN BY RECEIVER"
          assign to op_pattern.
      x. Ask for "ZONE PASS THROWN TO" assign to op_zone.
      xi. Ask for "DEPTH OF PASS THROWN" assign to op_depth.
    k. Ask for "YARDS GAINED OR LOST" assign to oresult.
    l. Ask the user if he wants to enter any more plays
    m. If the answer is no set done to true
    n. Call procedure build_o_list passing it t_ptr
6. Call procedure make_o_file
```

3 REED_OLDO_PLAYS

This procedure reads a file of plays that was build by an earlier run of the program.

Local variables
   j of type integer
   ch of type character

ALGORITHM:

```
  1. Reset the infile to the name contained in file_ino.

  2. While you have not reached the end of the file
    A. Read the information on the file into the proper
       variables and read off the spaces between the items.
    B. Call procedure build_o_list to put this information
       into the list of records of offensive plays.
```

4 OLIST_CHART

This procedure askes the user for the plays he wants included in the list charts and calls the procedure to have them printed.

```
Local variables
   wantedptr of type o_ptr
   okey,d_no,p_no,gl_key of type integer
   oh_key,z_key,rp_key,ohole_key,ows_key
   pz_key,another_o of type char
   dd_key,obcp_key,obc_no,rn_key,pd_key,
    of type two_string
   rp_key of type three_string
   oform_key,op_key,pa_key,pp_key of
    type eight_string
   do_o_again of type boolean
   wfile,typecharts,more_o,srted of
    type char
```

```
ALGORTHIM
  1. Repeat until all the required charts have
     been made. (do_o_again = false)
    A. Create a new play record with wantedptr pointing to it.
    B. Assign 999 to the result, drive_no, and play_no
       fields of the new record.
    C. Assign nil to the next_op field of the new record.
    D. Assign a not-selected flag to all other fields of
       the new record.
    E. Ask if the charts are to be written to a file or to
       the terminal. Assign the answer to wfile.
    F. If they are to a file then
      a. If file_set is false then
        i. Reset the outfile to file_out3.
        ii. Assign true to file_set.
      b. Assign true to write_to_ofile.
    G. Ask if the user wants the plays in the
       order entered or in gain-loss order.
    H. Assign the answer to srted.
    J. If the plays are to be in gain-loss
       order then
      a. Assign true to sorted.
      else
      b. Assign false to sorted.
    K. Ask the user how many keys he wants the
       charts keyed on. Assign the answer to
       numkeys.
    L. If numkeys equals 1 then
  a. Assign false to many_keys.
  else
  b. Assign true to many_keys.
M. Assign true to odd_key.
N. Assign 1 to which_key.
O. Assign true to first.
P. While numkeys is greater than zero do
  a. Call procedure init_field.
  b. If which key = 1 then
    i. Assign "FIRST" to request.
    else
    ii. Assign "NEXT " to request.
  c. Increment which_key.
  d. Call procedure init_field.
  e. Call procedure init_gain.
  f. Write to the terminal"WHAT DO YOU WANT THE"
     request"KEY TO BE"
      Give the user the following choices:
    1. DRIVE NUMBER               11. HOLE
    2. PLAY NUMBER                12. STRONG OR WEAK SIDE
    3. HASH MARK                  13. RESULT (GAIN OR LOSS)
    4. ZONE                       14. PASS ACTION
    5. DOWN & DISTANCE            15. POSITION of RECEIVER
    6. FORMATION                  16. NUMBER of RECEIVER
    7. RUN or PASS                17. PASS RESULT
    8. PLAY                       18. PASS PATTERN
    9. POSITION of BALL CARRIER   19. DEPTH of PASS
   10. NUMBER of BALL CARRIER     20. ZONE OF PASS
   21. ALL PLAYS
  g. Assign his choice to okey
  h. If okey = 1 then
    i. Assign "DRIVE NUMBER" to field.
    ii. Ask which drive number he wants plays for
    iii. Assign his answer to the drive_no field of
         the record that wantedptr points to.
  i. If okey = 2 then
    i. Assign "PLAY NUMBER" to field.
    ii. Ask which play number he wants plays for
    iii. Assign his answer to the play_no field of
         the record that wantedptr points to.
  j. If okey = 3 then
    i. Assign "HASH MARK" to field.
    ii. Ask which hash mark he wants plays for
    iii. Assign his answer to the hash field of
         the record that wantedptr points to.
  k. If okey = 4 then
    i. Assign "YARDAGE ZONE" to field.
    ii. Ask which yardage zone he wants plays for
  iii. Assign his answer to the zone field of
        the record that wantedptr points to.
l. If okey = 5 then
  i. Assign "DOWN and DISTANCE" to field.
  ii. Ask which down and distance he wants plays for
  iii. Assign his answer to the d_d field of
        the record that wantedptr points to.
m. If okey = 6 then
  i. Assign "OFFENSIVE FORMATION" to field.
  ii. Ask which formation he wants plays for
  iii. Assign his answer to the form field of
        the record that wantedptr points to.
n. If okey = 7 then
  i. Assign "RUN or PASS" to field.
  ii. Ask if runs or passes are wanted.
  iii. Assign his answer to the r_p field of
        the record that wantedptr points to.
o. If okey = 8 then
  i. Assign "THE PLAY RUN" to field.
  ii. Ask which play he wants plays for
  iii. Assign his answer to the play field of
        the record that wantedptr points to.
p. If okey = 9 then
  i. Assign "BALL CARRIER POS" to field.
  ii. Ask which ball carrier position he wants plays for
  iii. Assign his answer to the pos_bc field of
        the record that wantedptr points to.
q. If okey = 10 then
  i. Assign "BALL CARRIERS #" to field.
  ii. Ask which ball carrier he wants plays for
  iii. Assign his answer to the b_c field of
        the record that wantedptr points to.
r. If okey = 11 then
  i. Assign "HOLE NUMBER #" to field.
  ii. Ask which hole he wants plays for
  iii. Assign his answer to the hole field of
        the record that wantedptr points to.
s. If okey = 12 then
  i. Assign "STRONG OF WEAK SEDE" to field.
  ii. Ask which side he wants plays for
  iii. Assign his answer to the b_c field of
        the record that wantedptr points to.
t. If okey = 13 then
  i. Assign "YARS GAINED#" to field.
  ii. Ask which gain he wants plays for
  iii. Assign his answer to the result field of
        the record that wantedptr points to.
u. If okey = 14 then
```

```
    i. Assign "PASS ACTION" to field.
    ii. Ask which pass action he wants plays for
    iii. Assign his answer to the p_act field of
         the record that wantedptr points to.
  v. If okey = 15 then
    i. Assign "RECEIVER POSITION" to field.
    ii. Ask which receiver position he wants plays for
    iii. Assign his answer to the rec_p field of
         the record that wantedptr points to.
  w. If okey = 16 then
    i. Assign "RECEIVER NUMBER" to field.
    ii. Ask which receiver number he wants plays for
    iii. Assign his answer to the rec field of
         the record that wantedptr points to.
  x. If okey = 17 then
    i. Assign "PASS RESULT" to field.
    ii. Ask which pass result he wants plays for
    iii. Assign his answer to the p_reslt field of
         the record that wantedptr points to.
  y. If okey = 18 then
    i. Assign "PASS PATTERN" to field.
    ii. Ask which pass pattern he wants plays for
    iii. Assign his answer to the p_pattern field of
         the record that wantedptr points to.
  z. If okey = 19 then
    i. Assign "PASS DEPTH" to field.
    ii. Ask which pass depth he wants plays for
    iii. Assign his answer to the p_depth field of
         the record that wantedptr points to.
 aa. If okey = 20 then
    i. Assign "ZONE OF PASS" to field.
    ii. Ask which pass zone he wants plays for
    iii. Assign his answer to the p_zone field of
         the record that wantedptr points to.
 bb. If okey = 21 then
    i. Assign "ALL PLAYS" to field.
 cc. decrement numkeys.
Q. If the chart id to a file then
  a. Call procedure print_list_chart passing
     wantedptr and outfile.
   else
  b. Call procedure print_list_chart passing
     wantedptr and tty.
R. Assign false to o_headed.
S. Ask the user if he wants another list chart
T. assign the answer to another_o.
U. If his answer is yes then
  a. Assign true to do_o_again
```

5 PRINT_LIST_CHART(rp : o_ptr; out_dv :text)

This procedure writes the list of plays that match the plays that the user wants.

```
Local variables
  tp of type o_ptr.

ALGORITHM
  1. If the list is to be in gain loss order then
    A. Assign so_anchor to tp.
    else
    B. Assign o_anchor to tp.

  2. Call procedure o_heading passing out_dv.

  3. While tp is not nil do
    A. If each field of the record that rp points to
       is not-selected or it equals the same field of
       the record that tp points to then
      a. Increment gain indexed by the result field that
         tp points to.
      b. Increment numplays.
      c. Write to the output the fields of the record that
         tp points to separated by "|" in the following order.
         1. ot.
         2. drive_no.
         3. play_no.
         4. hash.
         5. zone.
         6. form.
         7. d_d.
         8. r_p.
         9. play.
        10. pos_bc.
        11. b_c.
        12. hole.
        13. s_w.
        14. p_act.
        15. rec_p
        16. rec.
        17. p_pattern.
        18. p_depth.
        19. pzone.
        20. p_reslt.
        21. result.
    d. Call procedure draw_line passing 79
       and out_dv.
  B. Assing the next_op field that tp points to
     to tp.

4. Call procedure o_trailer passing out_dv.

5. Write 3 blank lines to the output.
```

6 O_TRAILER(var out_dv : text)

This procedure prints the trailer to the offensive charts that are written to the terminal.

ALGORITHM
```
  1. Call function average and write what it
     returns to the output.

  2. Call function mode and write what it returns
     on the next line.

  3. Write "TOTAL NUMBER OF YARDS = " total
     on the next line.

  4. Call function median and write what it
     returns on the next line.

  5. If numplays is greater than 2 then
    A. Call function mean and write what it
       returns on the next line.
     else
    B. Write"TWO OR LESS PLAYS ON THIS CHART
       SO NO MEAN IS CALCULATED"

  6. Write "TOTAL NUMBER OF PLAYS ON THIS
     CHART = " numplays on the next line.

  7. Write the percent of all the plays that
     numplays is on the next line.

  8. Write "TOTAL PLAYS ENTERED = " total_oplays
     on the next line.

  9. Call function st_dev and write what it
     returns on the next line.
```

6.2 <u>MODE</u>

This is a function of type integer.

This function finds the most frequent gain or loss. If there are two or more that are equal it will find the lowest gain or loss value of the ones that are equal.

Local variables
 temp_mode, i of type integer.

ALGORITHM
```
  1. If numplays = 0 then
    A. Assign 0 to mode.
     else
    B. Assign the value of the gain[-100]
       to temp_mode.
    C. For i from -100 tp 100 do
      a. If gain[i] is greater than temp_mode then
        i. Assign gain[i] to temp_mode.
        ii. Assign i to mode
```

6.3 MEAN

This is a function of type real.

This function finds the average gain if the highest and lowest values are not counted.

Local variables
  i, highest,lowest of type integer
  lowfound of type boolean.

```
ALGORITHM
  1. Assign false to lowfound.

  2. For i assigned -100 to 100 do
    A. If lowfound is false then
      a. If gain[i] is greater then 0 then
        i. Assign i to lowest.
        ii. Assign true to lowfound
      b. If gain[i] is greater then 0 then
        i. Assign i to highest.

  3. Assign what ((average returns times numplays)
     minus (highest plus lowest)) all divided by
     (numplays minus two) to mean.
```

6.4 ST_DEV

This is a function of type real.

This function will find the standard deviation for the gain or loss values on the chart being produced.

```
Local variables
  i, j of type integer;
  sum, a of type real.

ALGORITHM
  1. If numplays = 0 then
    A. Assign 0 to st_dev.
     else
    B. Call function average and assign what it returns
       to a.
    C. Assign 0 to sum.
    D. For i assign -100 to 100
     a.  If gain[i] does not equal then
       i.  For j assigned 1 to gain[i] do
         ii. Assign sum plus the square of
             the absolute value of i - a to sum.
    E. Assign the square root of sum divided by
       numplays to st_dev.
```

6.5 MEDIAN

This is a function of type integer.

This function finds the middle gain or loss value for a chart of plays.

```
Local variables
  got_it  i of type integer;
  key of type real;
  done of type boolean.

ALGORITHM
  1. If numplays = 0 then
    A. Assign 0 to mode.
     else
    B. Assign -100 to i.
    C. Assign numplays divided by two to key.
    D. Assign 0 to got_it.
    E. Assign false to done.
    F. Repeat until done is true or i equals 100
      a. If gain[i] is greater then 0 then
        i. Assign got_it plus gain[i] to got_it.
      b. If got_it is greater than or equal
          to key then
        i. Assign i to median.
        ii. Assign true to done.
      c. Assign i plus 1 to i.
```

6.6 AVERAGE

This is a function of type real.

This function finds the average number of yards gained.

Local variable
  i of type integer.

ALGORITHM
```
  1. Assign 0 to total.

  2. For i assigned -100 to 100 do
    A. Assign total + (gain[i] * i) to total
    B. Assign total divided by numplays to average.
```

7 PRINT_DOWN_BY_PLAY_AR(var out_dv : text)

This procedure writes the down by play chart.

```
Local variables
  i of type integer
  d of type downs
  pl of type plays

ALGORITHM
  1. Write the title "     DOWN BY PLAY"
     to the output.

  2. Call procedure print_down_heading
     passing out_dv.

  3. Assign zero to 1.

  4. For pl from p59 to otherplay do
    A. Assign i + i to i.
    B. Write playnames[i] to the output.
    C. For d from ol to otherdown do
      a. Write down_by_play_ar[d,pl]
         to the output.
    D. Write tplay[i] and pplay[i]
       to the output.
    E. Call procedure draw_line passing
       74 and out_dv.

  5. Call procedure print_down_totals
     passing out_dv.
```

7.2 PRINT_DOWN_HEADING(var out_dv : text)

This procedure writes the heading on the down matrix charts.

```
ALGORITHM
  1. Write the following heading on one
     line to the output.
   "| 1L| 1M| 1S| 1G| 2L| 2M| 2S| 2G| 3L|
      3M| 3S| 3G| 4L| 4M| 4S| 4G|oth| # | %'".

  2. Call procedure draw_line passing
     79 and out_dv.
```

7.2 PRINT_FORMATION_HEADING(var out_dv : text)

This procedure prints the heading on the formation matrix charts.

```
ALGORITHM
  1. Write the following heading on one
     line to the output.
   "|PRI|PLI|PRQ|PLQ|SRI|SLI|SRQ|SLQ|SRS|
     SLS|PRA|PLA|PRB|PLB|TWR|TWL|oth| # | %'".

  2. Call procedure draw_line passing 79
     and out_dv.
```

7.2 PRINT_PASS_ROUTE_HEADING(var out_dv : text)

This procedure prints the heading on the pass route matrix charts.

ALGORITHM

```
1. Write the following heading on one
   line to the output.
  "| 1 | 2 | 3 |3SW| 4 |4SW| 5 |5SW|other".

2. Call procedure draw_line passing 42
   and out_dv.
```

7.2 PRINT_PASS_ACTION_HEADING(var out_dv : text)

This procedure prints the heading on the pass action matrix charts.

```
ALGORITHM
  1. Write the following heading on one
     line to the output.
   "|500|700|900| 80| 90|BOT|SCR|YSC|other".

  2. Call procedure draw_line passing 41
     and out_dv.
```

7.2 PRINT_HOLE_HEADING(var out_dv : text)

This procedure prints the heading on the hole matrix charts.

```
ALGORITHM
  1. Write the following heading on one
     line to the output.
   "| 9 | 7 | 5 | 3 | 1 | 0 | 2 | 4 | 6 |
      8 |PAS|other".

  2. Call procedure draw_line passing 54
     and out_dv.
```

7.3 PRINT_FORMATION_PERCENTS(var out_dv : text)

This procedure writes the percents for the columns in the formation matrix charts.

```
Local variable
  i of type integer

ALGORITHM
  1. Write four spaces to the output.

  2. For i from 1 to 17 do
    A. Write pformchart[i] to the output.

  3. Write " THERE ARE " pformchart[18] " PLAYS ON
       THIS CHART" to the output.
```

7.3 PRINT_PASS_ROUTE_PERCENTS(var out_dv : text)

This procedure writes the percents for the columns in the pass route matrix charts.

```
Local variable
  i of type integer

ALGORITHM
  1. Write four spaces to the output.

  2. For i from 1 to 9 do
    A. Write proutechart[i] to the output.

  3. Write " THERE ARE " proutechart[18] " PLAYS ON
     THIS CHART" to the output.
```

PRINT_PASS_ACTION_PERCENTS(var out_dv : text)

This procedure writes the percents for the columns in the pass action matrix charts.

```
Local variable
  i of type integer

ALGORITHM
  1. Write four spaces to the output.

  2. For i from 1 to 9 do
    A. Write ppactchart[i] to the output.

  3. Write " THERE ARE " ppactchart[18] " PLAYS ON
     THIS CHART" to the output.
```

7.3 PRINT_HOLE_PERCENTS(var out_dv : text)

This procedure writes the percents for the columns in the hole matrix charts.

```
Local variable
  i of type integer

ALGORITHM
  1. Write four spaces to the output.

  2. For i from 1 to 12 do
    A. Write pholechart[i] to the output.

  3. Write " THERE ARE " pholechart[18] " PLAYS ON
     THIS CHART" to the output.
```

7.3 PRINT_DOWN_PERCENTS(var out_dv : text)

This procedure writes the percents for the columns in the done matrix charts.

Local variable
  i of type integer

ALGORITHM

```
1. Write four spaces to the output.

2. For i from 1 to 17 do
  A. Write pdownchart[i] to the output.

3. Write " THERE ARE " pdownchart[18] " PLAYS ON
   THIS CHART" to the output.
```

7.4 PRINT_FORMATION_TOTALS(var out_dv : text)

This procedure prints the totals at the bottom of the columns for the formation matrix charts.

```
Local variables
  i of type integer

ALGORITHM
  1. Write four spaces to the output.

  2. For i from 1 to 17 do
    A. Write tformcharts[i] to the output.

  3. Call procedure print_formation_percents
     passing out_dv.

  4. Put two blank lines on the output.
```

7.4 PRINT_PASS_ROUTE_TOTALS(var out_dv : text)

This procedure prints the totals at the bottom of the columns for the pass route matrix charts.

```
Local variables
  i of type integer

ALGORITHM
  1. Write four spaces to the output.

  2. For i from 1 to 9 do
    A. Write troutecharts[i] to the output.

  3. Call procedure print_pass_route_percents
     passing out_dv.

  4. Put two blank lines on the output.
```

7.4 PRINT_PASS_ACTION_TOTALS(var out_dv : text)

This procedure prints the totals at the bottom of the columns for the pass action matrix charts.

```
Local variables
  i of type integer

ALGORITHM
  1. Write four spaces to the output.

  2. For i from 1 to 9 do
    A. Write tpactcharts[i] to the output.

  3. Call procedure print_pass_action_percents
     passing out_dv.

  4. Put two blank lines on the output.
```

7.4 PRINT_HOLE_TOTALS(var out_dv : text)

This procedure prints the totals at the bottom of the columns for the hole matrix charts.

Local variables
  i of type integer

```
ALGORITHM
  1. Write four spaces to the output.

  2. For i from 1 to 12 do
    A. Write tholecharts[i] to the output.

  3. Call procedure print_hole_percents
     passing out_dv.

  4. Put two blank lines on the output.
```

7.4 PRINT_DOWN_TOTALS(var out_dv : text)

This procedure prints the totals at the bottom of the columns for the down matrix charts.

Local variables
  i of type integer

```
ALGORITHM
  1. Write four spaces to the output.

  2. For i from 1 to 17 do
    A. Write tdowncharts[i] to the output.

  3. Call procedure print_down_percents
     passing out_dv.

  4. Put two blank lines on the output.
```

8 PRINT_DOWN_BY_FORMATION_AR(var out_dv:text)

This procedure writes the down by formation matrix chart.

```
Local variables
  i of type integer
  ho of tyoe holes
  f of type formations

ALGORITHM
  1. Write the title "     HOLE BY FORMATION"
     to the output.

  2. Call procedure print_hole_heading
     passing out_dv.

  3. Assign zero to 1.

  4. For f from pri to otherform do
    A. Assign i + i to i.
    B. Write formationnames[i] to the output.
    C. For ho from hnine to otherhole do
      a. Write hole_by_formation_ar[ho,f]
         to the output.
    E. Call procedure draw_line passing
       74 and out_dv.

  5. Call procedure print_hole_totals
     passing out_dv.
```

9 <u>PRINT_HOLE_BY_DOWN_AR(var out_dv:text)</u>

This procedure writes the hole by down chart.

```
Local variables
  i of type integer
  d of type downs
  ho of type plays

ALGORITHM
  1. Write the title "     HOLE BY DOWN"
     to the output.

  2. Call procedure print_hole_heading
     passing out_dv.

  3. Assign zero to 1.

  4. For d from ol to otherdown do
    A. Assign i + i to i.
    B. Write downnames[i] to the output.
    C. For ho from hnine to otherhole do
      a. Write hole_by_down_ar[ho,d]
         to the output.
    E. Call procedure draw_line passing
       54 and out_dv.

  5. Call procedure print_hole_totals
     passing out_dv.
```

10 PRINT_HOLE_BY_HASH_AR(var out_dv:text)

This procedure writes the hole by hash chart.

```
Local variables
  i of type integer
  hm of type hashmarks
  ho of type plays

ALGORITHM
  1. Write the title "     HOLE BY HASH"
     to the output.

  2. Call procedure print_hole_heading
     passing out_dv.

  3. Assign zero to 1.

  4. For hm from right to otherhash do
    A. Assign i + i to i.
    B. Write hashnames[i] to the output.
    C. For ho from hnine to otherhole do
      a. Write hole_by_hash_ar[ho,hm]
         to the output.
    E. Call procedure draw_line passing
       54 and out_dv.

  5. Call procedure print_hole_totals
     passing out_dv.
```

11 PRINT_PASS_ACTION_BY_HASH_AR(var out_dv:text)

This procedure writes the pass action by hash chart.

```
Local variables
  i of type integer
  pa of type pass_actions
  hm of type hashmarks

ALGORITHM
  1. Write the title "     PASS ACTION BY HASH"
     to the output.

  2. Call procedure print_pass_action_heading
     passing out_dv.

  3. Assign zero to 1.

  4. For hm from right to otherhash do
    A. Assign i + i to i.
    B. Write hashnames[i] to the output.
    C. For pa from a500 to otherpassaction do
      a. Write pact_by_hash_ar[pa,hm]
         to the output.
    E. Call procedure draw_line passing
       41 and out_dv.

  5. Call procedure print_pass_action_totals
     passing out_dv.
```

12 PRINT_PASS_ROUTE_BY_HASH_AR(var out_dv:text)

This procedure writes the pass route by hash chart.

```
Local variables
  i of type integer
  r of type routes
  hm of type hashmarks

ALGORITHM
  1. Write the title "     PASS ROUTE BY HASH"
     to the output.

  2. Call procedure print_pass_route_heading
     passing out_dv.

  3. Assign zero to 1.

  4. For hm from right to ptherhash do
    A. Assign i + i to i.
    B. Write hashnames[i] to the output.
    C. For r from r1 to otherroute do
      a. Write route_by_hash_ar[r,hm]
         to the output.
    E. Call procedure draw_line passing
       42 and out_dv.

  5. Call procedure print_hole_totals
     passing out_dv.
```

13 PRINT_FORMATION_BY_HASH_AR(var out_dv:text)

This procedure writes the formation by hash chart.

```
Local variables
  i of type integer
  f of type formations
  hm of type hashmarks

ALGORITHM
  1. Write the title "     FORMATION BY HASH"
     to the output.

  2. Call procedure print_formation_heading
     passing out_dv.

  3. Assign zero to 1.

  4. For hm from right to otherhash do
    A. Assign i + i to i.
    B. Write hashnames[i] to the output.
    C. For f from pri to otherform do
      a. Write form_by_hash_ar[f,hm]
         to the output.
    E. Call procedure draw_line passing
       74 and out_dv.

  5. Call procedure print_formation_totals
     passing out_dv.
```

14 PRINT_PASS_ROUTE_BY_ZONE_AR(var out_dv:text)

This procedure writes the pass route by zone chart.

```
Local variables
  i of type integer
  r of type routes
  z of type zones

ALGORITHM
  1. Write the title "     PASS ROUTE BY ZONE"
     to the output.

  2. Call procedure print_pass_route_heading
     passing out_dv.

  3. Assign zero to 1.

  4. For z from zone to otherzone do
    A. Assign i + i to i.
    B. Write zonenames[i] to the output.
    C. For r from r1 to otherroute do
      a. Write route_by_zone_ar[r,z]
         to the output.
    E. Call procedure draw_line passing
       42 and out_dv.

  5. Call procedure print_pass_route_totals
     passing out_dv.
```

15 PRINT_PASS_ACTION_BY_ZONE_AR(var out_dv : text)

This procedure writes the pass route by zone chart.

```
Local variables
  i of type integer
  pa of type pass_actions
  z of type zones

ALGORITHM
  1. Write the title "     PASS ACTION BY ZONE"
     to the output.

  2. Call procedure print_pass_action_heading
     passing out_dv.

  3. Assign zero to 1.

  4. For z from zone to otherzone do
    A. Assign i + i to i.
    B. Write zonenames[i] to the output.
    C. For pa from a500 to otherpassaction do
      a. Write pact_by_zone_ar[pa,z]
         to the output.
    E. Call procedure draw_line passing
       41 and out_dv.

  5. Call procedure print_pass_action-totals
     passing out_dv.
```

16 PRINT_PASS_ACTION_BY_DOWN_AR(var out_dv : text)

This procedure writes the pass action by down chart.

```
Local variables
  i of type integer
  pa of type pass_actions
  d of type downs

ALGORITHM
  1. Write the title "     PASS ACTION BY DOWN"
     to the output.

  2. Call procedure print_pass_action_heading
     passing out_dv.

  3. Assign zero to 1.

  4. For d from ol to otherdown do
    A. Assign i + i to i.
    B. Write donenames[i] to the output.
    C. For pa from a500 to otherpassaction do
      a. Write pact_by_down_ar[pa,d]
         to the output.
    E. Call procedure draw_line passing
       41 and out_dv.

  5. Call procedure print_pass_action_totals
     passing out_dv.
```

17 <u>PRINT_PASS_ROUTE_BY_DOWN_AR(var out_dv : text)</u>

This procedure writes the pass route by down chart.

```
Local variables
  i of type integer
  r of type routes
  d of type downs

ALGORITHM
  1. Write the title "     PASS ROUTE BY DOWN"
     to the output.

  2. Call procedure print_pass_route_heading
     passing out_dv.

  3. Assign zero to 1.

  4. For d from ol to otherdown do
    A. Assign i + i to i.
    B. Write downnames[i] to the output.
    C. For r from r1 to otherroute do
      a. Write route_by_down_ar[r,d]
         to the output.
    E. Call procedure draw_line passing
       42 and out_dv.

  5. Call procedure print_pass_route_totals
     passing out_dv.
```

18 PRINT_FORMATION_BY_DOWN_AR(var out_dv : text)

This procedure writes the formation by down chart.

```
Local variables
  i of type integer
  f of type formations
  d of type downs

ALGORITHM
  1. Write the title "     FORMATION BY DOWN"
     to the output.

  2. Call procedure print_formation_heading
     passing out_dv.

  3. Assign zero to 1.

  4. For d from ol to otherdown do
    A. Assign i + i to i.
    B. Write downnames[i] to the output.
    C. For f from pri to otherform do
      a. Write form_by_down_ar[f,d]
         to the output.
    E. Call procedure draw_line passing
       74 and out_dv.

  5. Call procedure print_formation_totals
     passing out_dv.
```

19 PRINT_FORMATION_BY_PLAY_AR(var out_dv : text)

This procedure writes the formation by play chart.

```
Local variables
  i of type integer
  f of type formations
  pl of type plays

ALGORITHM
  1. Write the title "     FORMATION BY PLAY"
     to the output.

  2. Call procedure print_formation_heading
     passing out_dv.

  3. Assign zero to 1.

  4. For pl from p59 to otherplay do
    A. Assign i + i to i.
    B. Write playnames[i] to the output.
    C. For f from pri to otherform do
      a. Write form_by_play_ar[f,pl]
         to the output.
    E. Call procedure draw_line passing
       74 and out_dv.

  5. Call procedure print_formation_totals
     passing out_dv.
```

20 INIT_FIELD

This procedure initializes the array that gives the charts thier headings to spaces.

```
Local variables
  i of type integer.

ALGORITHM
  1. For i assigned 1 to 20 do
    A. Assign a space to field[i].
```

21 MAKE_O_FILE

This procedure writes the records of the plays entered into a file so that the can be used again in another run of the program.

Local variable
  l_ptr of type o_ptr

ALGORITHM

```
  1. Rewrite the outfile to the name in file_out1.

  2. Assign o_anchor to l_ptr.

  3. Repeat until l_ptr is nil.
    A. Write to the outfile on one line the values in the
       the record that l_ptr points to in the following
       order, each value is to be followed by a space.
      1. opponent
      2. drive_no (in a field of 2)
      3. play_no (in a field of 2)
      4. result (in a field of 2)
      5. hash
      6. zone
      7. d_d
      8. form
      9. r_p
     10. play
     11. pos_bc
     12. b_c
     13. hole
     14. s_w
     15. p_act
     16. p_reslt
     17. rec_p
     18. rec
     19. p_pattern
     20. p-depth
     21. p_zone
    B. Assign the pointer to the next offensive record
       to l_ptr.
```

22 BUILD_O_LIST(var temp_ptr : o_ptr)

This procedure creates a linked list of offensive play records in the order that they are entered.

ALGORITHM:

```
 1. Create a pointer to a new offensive play record.

 2. Check to see if this is the first record to be
    built. If it is then
   A. Assign it to the anchor pointer.
   B. Assign it to the temp_ptr.
  else
   C. Assign the new offensive record to the next_op
      field of the last record.
   D. Assign the new record to the temp_ptr.

 3. Assign each of the values read into the offensive
    information variables to the proper field in the
    new record.

 4. Assign nil to the next play field of the record.

 5. Call procedure sorted_list passing the value of the
    result field of the new record.
```

23 SORTED_LIST(GAIN : INTEGER)

This procedure builds a list of the plays in ascending order by number of yards gained.

```
Local variables
   prev of type o_ptr to point to the last record looked at
   pres of type o_ptr to point to the record being looked at
   done of type boolean
   bigger of type boolean

ALGORITHM
  1. Initialize prev and pres to nil.

  2. Initialize done and bigger to false.

  3. Create a new offensive record with so_current pointing to it.

  4. If the sorted list is empty then
    A. Assign the new record to
       so_anchor.
     else
    B. Assign pres to what so_anchor points to.

  5. If gain is less than the result field of the record
     that pres points to then
    A. Assign so_current to so_anchor.
    B. Assign what pres is pointing to to the next_op
       field of the new record.
      else
    C. Repeat until done is true
      a. If the next_op field that pres points to is nil or
         gain is less than the result field that pres points
         to then
        i. Assign true to done.
      else
        ii. Assign pres to prev.
        iii. Assign the next_op field of what pres is pointing
             to to pres.
    D. If gain is less than the result field that pres
       is pointing to then
      a. Assign true to bigger.
    E. If bigger is true then
      a. Assign the record that so_current points to to
         the next_op field of the record that prev
         points to.
      b. Assign pres to the next_op field of the record
```

```
       so_current points to.
     else
    c. Put the record that so_current points to at the.
      end of the list.

6. Assign all the offensive play elements read in to the
   proper fields of the record that so_current points to
```

24 O_HEADING(var out_dv : text)

ALGORITHM:

1. Print the following heading to the output.

```
|  |  | | |      | d|r|      |  |  | | |        | |r |       |  | |  |  |
|dn| n| | |      | i|u|      |p |n | | |        |r|en|.      |  |p| r|  |
|ru|pu|h|z|      |ds|n|      |o |u |h|s|        |e|cu|       | d|!|pe|gl|
|im|lm|a|o|      |ot|!|      |s |mb|o| |        |c|em|   p   | e|z|as|ao|
|vb|ab|s|n| form |wa|p| play |! |bc|l|!|p-action|!|ib|       |pp|o|su|is|
|ee|ye|h|e|      |nn|a|      |b |e |e| |        |p|ve|pattern| t|n|sl|ns|
| r| r| | |      | c|s|      |c |r | |w|        |o|er|       | h|e| t|  |
|  |  | | |      | e|s|      |  |  | | |        |s|r |       |  | |  |  |
```

25 <u>DRAW_LINE(length : integer;var out_dv : text)</u>

This procedure writes lines on the output charts.

Local variable
  i of type integer

ALGORITHM
```
  1. For i assigned 1 to length
    A. Print a dash to the output.
```

26 INIT_GAIN

This procedure initilizes the array that holds the number of plays for each gain and loss value.

Local variables
   i of type integer to be used as a loop control
      and an index counter

ALGORITHM

```
1. Assign 0 to each element of gain.

2. Assign 0 to numplays.
```

27 INIT_FILENAME

This procedure initializes the input and output filenames to spaces.

Local variable
  i of type integer

ALGORITHM:
1. Assign space to each of the elements of file_out1.
2. Assign file_out1 to file_out2.
3. Assign file_out1 to file_out3.
4. Assign file_out1 to file_out4.
5. Assign file_out1 to file_ino.
6. Assign file_out1 to file_ind.

28 FILENAME

This procedure assigns the proper names to the variables to be used as file names by the program.

```
Local variable
  i or type integer

ALGORITHM
  1. Assign file_out1 to file_out2.

  2. Assign file_out1 to file_out3.

  3. Assign file_out1 to file_out4.

  4. Assign 1 to i.

  5. Repeat until file_out1 indexed by i is a space
    A. Assign i+1 to i.

  6. Assign a period to file_out1 indexed by i.

  7. Assign a period to file_out2 indexed by i.

  8. Assign a period to file_out3 indexed by i.

  9. Assign a period to file_out4 indexed by i.

 10. Assign a O to file_out1 indexed by i+1.

 11. Assign a D to file_out2 indexed by i+1.

 12. Assign a O to file_out3 indexed by i+1.

 13. Assign a D to file_out4 indexed by i+1.

 14. Assign a C to file_out3 indexed by i+2.

 15. Assign a C to file_out4 indexed by i+2.

 16. Assign each element of file_out1 to file_ino.

 17. Assign each element of file_out1 to file_ind.
```

USER'S MANUAL

for

```
     d       SSSSS    CCCCC    OOOOO    U     U   TTTTTTT
     d      S        C        O     O   U     U      T
     d      S        C        O     O   U     U      T
 ddddd ===   SSSS    C        O     O   U     U      T
d    d           S   C        O     O   U     U      T
d    d           S   C        O     O   U     U      T
 ddddd      SSSSS     CCCCC    OOOOO     UUUUU       T
```

by K. Garry Dyer

d-SCOUT

Scout is designed to be a user friendly, work saving, football scouting program for any coaching staff with a computer capable of using Pascal. This program will take specific input and sort and organize it into several different charts which may be shown on the terminal or written to a file to be printed on hard copy. The input from each session is either saved in a file of its own or added to an existing file so that it can be used at a later date.

The files containing the input information and the charts to be printed will be named in the following manner: The file containing the information put in about a team's defense will be named by the name of the team being scouted followed by a period and the letter D e.g. BOBCATS.D. The files containing the charts to be printed will have the same names with the last letter followed by a C e.g. BOBCATS.DC.

When the program first starts the following display will appear on the screen.

```
     d       SSSSS   CCCCC    OOOOO   U     U  TTTTTTT
     d      S       C     C  O     O  U     U     T
     d      S       C        O     O  U     U     T
 ddddd ====  SSSS   C        O     O  U     U     T
d    d           S  C        O     O  U     U     T
d    d           S  C     C  O     O  U     U     T
 ddddd      SSSSS    CCCCC    OOOOO    UUUUU      T

                                   by K. Garry Dyer
```

```
press RETURN to start program
```

To start the main portion of the program you need to press the "RETURN" key. Upon doing this you will be asked a series of questions to determine exactly what use you want to make of the program with this run. Please note that all responses must be followed by a carriage return and that if a mistake is made in a response, it may be corrected before the carriage return. These questions and proper type responses are as follows:

GETTING STARTED

WHAT TEAM IS BEING SCOUTED?

The response to this question can be any set of up to fifteen characters and will be the name attached to the files that are generated for the purpose of scouting the team in question.

DO YOU WANT TO INCLUDE A PREVIOUSLY BUILT FILE OF PLAYS?

If you have a file on the team that you are scouting and want the information that is in that file to be included in this report you should respond with "Y" and the old file will be read into the data base for this run. If you have a file on this team but do not want it included in this set of reports you should be sure to name the team being scouted by a different name than was used for the other file. If you use the same name and do not include it in this run, the old file will be destroyed and the new one by that name will contain only the information entered during this run. If you do not have an old file of defensive information that you want included in this run you should respond with "N".

Next you will be asked if you want to enter new plays into the data base.

DO YOU WANT TO ADD MORE PLAYS TO THE REPORT? (Y/N)

If you wish to add more plays to an old data base or start a new one you should respond "Y" to this question. If you only want to generate some reports from an old file of plays and not add any new ones respond with "N". A response of "Y" will take you to the part of the program that accepts information about new plays.

ENTERING DEFENSIVE PLAYS.

To get the information about defensive plays you will be asked the following questions:

WHO IS THE OPPONENT FOR THIS SET OF PLAYS

This is the opponent of the team being scouted and the answer should be no more than three characters long.

HASH MARK L, M, OR R?

This is to indicate which hash mark the play started from. You should enter "R" for the right hash mark, "M" for the middle of the field, and "L" for the left hash mark.

YARDAGE ZONE 1 TO 6?

This is the zone on the field that the play starts from and can be any number from 1 to 6. The user may define these zones in whatever way works best for him. In this program these zones are set up as follows: 1: goal to 10, 2: 10 to 30, 3: 30 to 50, 4: 50 to 30, 5: 30 to 10, and 6: 10 to goal. The order of the zones is structured so that the offense moves toward a higher numbered zone. You should enter a number from 1 to 6 to indicate where the play started.

DOWN -&- DISTANCE?

This is the down and distance of the play. You should enter a number followed by a letter: 1 for first down, 2 for second down, 3 for third down, 4 for forth down. L for long-more than seven yards, M for medium-from 3 to 7 yards, S for short-less than 3 yards, and G-if it is less than first down distance to the goal line e.g. 1L or 3M.

OFFENSIVE FORMATION?

This is to indicate the offensive formation of the opponent of the team being scouted. This may be any group of characters up to eight. Be sure to use the same characters to indicate the same thing each time e.g. PRO-R-I, SHOTGUN, or VEER.

OFFENSIVE MOTION?

Enter any offensive motion that the opponent used. this can contain a maxium of three characters.

DEFENSIVE FRONT?

This is to indicate the defensive front that the team used. This may be any group of characters up to eight. Be sure to use the same characters to indicate the same thing each time e.g. 3-4, stack-6, gap-6, etc.

VARAITION TO FRONT

Enter any variation to the normal front that the team being scouted used. A maximum of three characters can be used.

DEFENSIVE COVERAGE?

This is to indicate the coverage that the secondary is using. This may be any group of characters up to eight. Be sure to use the same characters to indicate the same thing each time e.g. ZONE, MAN, ZONE-R etc.

DEFENSIVE STUNT?

This is to be used to enter any stunt that the offensive line or linebackers may have used. This may be any group of characters up to eight. Be sure to use the same characters to indicate the same thing each time.

BLITZ?

This is to enter any type of a blitz that the defense may have used. This may be any group of characters up to eight. Be sure to use the same characters to indicate the same thing each time e.g. L-OLB, ILB, etc.

This is the end of the defensive information. You will be asked if this is the last play that you want to enter. If you have more defensive plays to enter type "N" to return to the defensive play entry part of the program. If you have no more plays to enter type "Y" and you will exit to the part of the program that lets you have reports generated.

DEFENSIVE CHARTS

After all the defensive play information has been entered into the data base and you are finished entering defensive information, you will be given the choice to have defensive charts generated.

```
DO YOU WANT DEFENSIVE CHARTS?  (Y/N)
```

At this point if you want defensive charts you should type "Y", if not type "N". If you choose "N" you will be given a choice to terminate the run or to start over and enter more information or get different charts.

```
DO YOU WANT LIST CHARTS OR MATRIX CHARTS?

             L for LIST CHARTS

             M for MATRIX CHARTS

Please enter the letter to indicate which type of chart you wish.
```

If you choose to have defensive charts you will be given the choice of having them written to a file for printing on paper or to your terminal for instant inspection.

```
DO YOU WANT THE CHARTS WRITTEN TO A FILE TO BE PRINTED
OR TO THE TERMINAL?    T/F
```

If you want the charts to go to a file to be printed on paper you should type "F" if you want them to be written to your terminal type "T".

If you choose list charts you will now be asked how many items you want the chart keyed on.

HOW MANY ITEMS DO YOU WANT THE CHART KEYED ON?

You must now decide how many items (up to 10) that you want the chart keyed on. Type the number of keys that you want and you will be asked for the first key.

```
WHAT DO YOU WANT THE FIRST KEY TO BE?
TYPE THE APPROPRIATE NUMBER TO INDICATE THE KEY AREA
   1:  HASH MARK                     5: DEFENSIVE FRONT
   2:  DOWN and DISTANCE             6: SECONDARY COVERAGE
   3:  YARDAGE ZONE                  7: STUNTS
   4:  THE OFFENSIVE FORMATION       8: BLITZES
   9:  ALL PLAYS                    10: MOTION
```

You should now decide what report(s) you want and enter the correct number. Next you will be askedone of the nine questions that follows to find out what you want the value of key to be:

```
WHICH HASH MARK DO YOU WANT THE CHART FOR?

WHAT YARDAGE ZONE DO YOU WANT THE CHART FOR?

WHICH DOWN AND DISTANCE DO YOU WANT THE CHART FOR?

WHICH OFFENSIVE FORMATION DO YOU WANT THE CHART FOR?

WHAT DEFENSIVE FRONT DO YOU WANT THE CHART FOR?

WHAT COVERAGE DO YOU WANT CHARTS FOR?

WHAT STUNT DO YOU WANT THE CHART FOR?

WHAT BLITZ DO YOU WANT THE CHART FOR?

WHICH MOTION DO YOU WANT THE CHART FOR?
```

You must answer these questions with the exact same thing with which you entered the information. You will continue to be asked for the value of the keys until you have entered the value for each of the keys that you wanted for this chart. The chart that you have requested will now be written to a file or onto your terminal, depending on where you asked for it to be sent.

After the program has built the chart that you have requested, you will be given a chance to go back and request another chart.

DO YOU WANT ANOTHER LIST CHART?

Answer either "Y" or "N" depending on whether you want another list chart. If you answer "Y" the program will return to the above lines and let you make another list chart. If you answer "N" you will be given the chance to have matrix charts built.

DO YOU WANT ANOTHER DEFENSIVE CHART? Y/N

If you want another chart type "Y" and the program will go back and let you choose another chart. All of the charts from this run will be put into the same file, so that they may all be printed at once. When you have all of the defensive charts that you want, type "N" and the program will ask you if you are finished or if you want to run it longer.

DO YOU WANT ANOTHER RUN? Y/N

If you want to do another run for entering more plays or getting more charts type "Y" to get back to the start of the program. If you are finished for now type "N" and the program will stop.

If you choose to have matrix charts you will be given the following menu from which to choose the charts:

1 for COVERAGE BY FRONT

2 for COVERAGE BY FORMATION

3 for COVERAGE BY ZONE

4 for COVERAGE BY HASH MARK

5 for COVERAGE BY DOWN AND DISTANCE

6 for FRONT BY FORMATION

7 for FRONT BY DOWN AND DISTANCE

8 for FRONT BY ZONE

9 for FRONT BY HASH MARK

10 for ALL OF THE ABOVE CHARTS

You should enter the number that indicates the type of matrix chart you want created. After the chart has been created you will be given the same choices as before. You can either have more charts created or stop the program as explained above.

USER'S MANUAL

for

```
             SSSSS    CCCCC    OOOOO    U     U  TTTTTTT
            S        C        O     O   U     U     T
            S        C        O     O   U     U     T
 oooo  ===   SSSS    C        O     O   U     U     T
o    o           S   C        O     O   U     U     T
o    o           S   C        O     O   U     U     T
 oooo       SSSSS     CCCCC    OOOOO     UUUUU      T
```

by K. Garry Dyer

o-SCOUT

Scout is designed to be a user friendly, work saving, football scouting program for any coaching staff with a computer capable of using Pascal. This program will take specific input and sort and organize it into several different charts which may be shown on the terminal or written to a file to be printed on hard copy. The input from each session is either saved in a file of its own or added to an existing file so that it can be used at a later date.

The files containing the input information and the charts to be printed will be named in the following manner. The file containing the information put in about a team's offense will be named by the name of the team being scouted followed by a period and the letter O e.g. BOBCATS.O. The files containing the charts to be printed will have the same names with the last letter followed by a C e.g. BOBCATS.OC.

When the program first starts the following display will appear on the screen.

```
             SSSSS    CCCCC    OOOOO   U     U  TTTTTTT
            S        C     C  O     O  U     U     T
            S        C        O     O  U     U     T
 oooo  ====  SSSS    C        O     O  U     U     T
o    o           S   C        O     O  U     U     T
o    o           S   C     C  O     O  U     U     T
 oooo       SSSSS    CCCCC    OOOOO    UUUUU       T

                                    by K. Garry Dyer



   press RETURN to start program
```

To start the main portion of the program you need to press the "RETURN" key. Upon doing this you will be asked a series of questions to determine exactly what use you want to make of the program with this run. Please note that all responses must be followed by a carriage return and that if a mistake is made in a response, it may be corrected before the carriage return. These questions and proper type responses are as follows:

GETTING STARTED

WHAT TEAM IS BEING SCOUTED?

The response to this question can be any set of up to fifteen characters and will be the name attached to the files that are generated for the purpose of scouting the team in question.

DO YOU WANT TO INCLUDE A PREVIOUSLY BUILT FILE OF PLAYS?

If you have a file on the team that you are scouting and want the information that is in that file to be included in this report you should respond with "Y" and the old file will be read into the data base for this run. If you have a file on this team but do not want it included in this set of reports you should be sure to name the team being scouted by a different name than was used for the other file. If you use the same name and do not include it in this run, the old file will be destroyed and the new one by that name will contain only the information entered during this run. If you do not have an old file of offensive information that you want included in this run you should respond with "N".

Next you will be asked if you want to enter new plays into the data base.

DO YOU WANT TO ADD MORE PLAYS TO THE REPORT? (Y/N)

If you wish to add more plays to an old data base or start a new one you should respond "Y" to this question. If you only want to generate some reports from an old file of plays and not add any new ones respond with "N". A response of "Y" will take you to the part of the program that accepts information about new plays.

ENTERING NEW OFFENSIVE PLAYS

To allow for the entering of data about plays not all ready in the data base, the program will ask you the following list of questions. The order that plays are entered will be the relative order that they will appear, so it is best to think of how you want them organized before entering the information. You will be given the choice of having the reports in this order or in ascending order according to the gain or loss on the plays on the chart.

WHO IS THE OPPONENT?

This will put the opponent on the charts so the user will be able to distinguish one game from another if several games are on the same report. This will only be asked once for each set of plays entered, therefore if more than one game is being entered during one session you should start the program over for each new opponent.

DRIVE NUMBER?

You should answer this with the number of the possession for which you are entering plays. This can be any number from 1 to 99.

PLAY NUMBER?

This is the number of the play in the current possession and can be any number from 1 to 99.

HASH MARK-- R, M, L?

This is to indicate from which hash mark the play started You should enter "R" for the right hash mark, "M" for the middle of the field, and "L" for the left hash mark.

YARDAGE ZONE 1 TO 6?

This is the zone on the field that the play starts from and can be any number from 1 to 6. The user may define these zones in whatever way works best for him. In this program these zones are set up as follows: 1: goal to 10, 2: 10 to 30, 3: 30 to 50, 4: 50 to 30, 5: 30 to 10, and 6: 10 to goal. The order of the zones is so structured so that the offense moves toward a higher numbered zone. You should enter a number from 1 to 6 to indicate where the play started.

DOWN -&- DISTANCE?

This is the down and distance of the play. You should enter a number followed by a letter: 1 for first down, 2 for second down, 3 for third down, 4 for fourth down, L for long-more than seven yards, M for medium-from 3 to 7 yards, S for short-less than 3 yards, and G-if it is less than first down distance to the goal line e.g. 1L or 3M.

FORMATION?

In response to this, you should enter the formation in which the team set up. This can be any description that you choose up to 8 characters. Care should be taken always to use the exact same description for the same formation e.g. PRO-R-I, SHOTGUN, or VEER.

RUN or PASS   R or P? ENTER K FOR PUNT

ENTER F FOR FIELD GOAL

This is to enter whether the play was a run or a pass. For running plays enter an "R" , a "P" for passes, a "K" for punts, and a "F" for field goals.

PLAY?

This is where you enter the actual name or number of the play that was run. You can use any combination of letters, numbers, dashes etc. up to eight(8) characters. Be sure to use the exact same name or number for the same play.

The next four questions pertain only to running plays and will not be asked if you answered "P" to RUN or PASS.

THE POSITION OF THE BALL CARRIER?

The response to this is to be a letter or number indicating the position of the player who carried the ball. You may use any numbering scheme that you want e.g. 1 for quarterback, 2 for left halfback, 3 for right halfback, 4 for fullback, etc.

BALL CARRIERS NUMBER?

This is where you enter the actual number worn by the player that carried the ball.

HOLE NUMBER

This is to be the hole to whitch the ball carrier ran, the numbering scheme for this program is as shown below.

```
9      7      5    3   1  0  2   4   6       8
             0      0   0   0   0   0   0
                        0                 0

                    0       0
```

STRONG OR WEAK SIDE?

This question is asked for both the running plays and the passes. This is to indicate if the ball carrier ran or the pass was thrown to the strong or weak side of the formation. Enter "S" for the strong side, "W" for the weak side. If the formation does not have strong and weak sides or if the ball was carried up the center (to neither side) you may enter either "N" for neither or just enter a dash "-". Be sure to use the same character each time.

enter FUM if fumble on running play

If a fumble occurred on the play enter FUM at this time. If no fumble, occurred please enter a dash "-".

The following six questions apply only to passing plays and will not be asked if you answered "R" to RUN or PASS.

PASS ACTION?

This is used to describe the action of the quarterback or other passer on a pass play. You may enter any description up to eight characters. Be sure to use the same characters each time you describe the same type of action e.g. scramble, 5-drop, sprint-l. etc.

PASS RESULT--COM, INC, or INT?

This will let the user know if the pass was caught, dropped, or intercepted.

POSITION OF INTENDED RECEIVER?

The response to this is to be a number to indicate the position from which the intended pass receiver started. You may use any numbering scheme that works for you. Any number from 1 to 99 will work. Be sure to use the same numbering scheme for the entire program.

INTENDED RECEIVERS NUMBER?

This is to be the actual number worn by the intended receiver.

PATTERN RUN BY RECEIVER?

The response to this can be any combination of characters, It must not be more than eight characters long. Be sure to use the same characters each time you describe the same pattern e.g. Z-OUT, FLY, CROSS.

ZONE OF PASS

This is to be a number of one or two characters to indicate to which pass zone the ball was thrown. You may use any numbering scheme that works best for you.

The last two question will be asked for both run and pass. They will be asked if "K" for punt or "F" for field goal were answered to RUN or PASS.

DEPTH OF PASS THE THROWN?

VS=0-5, S=5-10, M=10-20, L=20-30, VL=30+

ENTER YARDAGE IF PUNT OR FIELD GOAL

If the play is a pass, this is to be the indicated depth area that the pass actually traveled in the air and is not to include any yards that were gained or lost by the running of the receiver after he caught the ball. If the play was a punt or a field goal the distance of the punt or field goal may be entered. The distance of the punt or field goal may be actual yards and does not need to follow the pass depth zoning scheme. You may want to indicate that a field goal was missed by entering the yardage of the attempt as a minus number.

YARDS GAINED OR LOST?

This is to be the actual number of yards that were gained or lost on the play. It may be any number from -99 to 99.

This is the end of entering a play. You will now be asked if this was the last play you wish to enter. A response of "N" will take you back to the start and allow you to enter another play. A response of "Y" will exit you from this part of the program and put you into the part that generates output charts.

OFFENSIVE CHARTS

If you selected offense and you are finished entering plays or did not want to enter any new plays you will be sent to this portion of the program.

There are two types of offensive charts that can be generated, List charts, and Matrix charts. A sample of each is shown below:

```
LIST CHART

CHART KEYED ON HASH MARK


|  |   | | |           | d|r|         |  |  | | |         | |r |          |  | |   |  |
|dn|  n| | |           | i|u|         |p |n | | |         |r|en|          |  |p|  r|  |
|ru| pu|h|z|           |ds|n|         |o |u |h|s|         |e|cu|          | d|!|p e|gl|
|im| lm|a|o|           |ot|!|         |s |mb|o| |         |c|em|    p     | e|z|a s|ao|
|vb| ab|s|n|   form    |wa|p|  play   |! |bc|l|!|p_action |!|ib|          |pp|o|s u|is|
|ee| ye|h|e|           |nn|a|         |b |e |e| |         |p|ve|pattern   | t|n|s l|ns|
| r|  r| | |           | c|s|         |c |r | |w|         |o|er|          | h|e|  t|  |
|  |   | | |           |&e|s|         |  |  | | |         |s|r |          |  | |   |  |
---------------------------------------------------------------------------------------
| 1|  2|R|3|T-T        |2S|R|SWEEP-L  |3 |44|8|S|-        |-|- |-         |- |-|FUM|-2|
---------------------------------------------------------------------------------------
| 3|  3|R|6|PRO-L-T    |1G|R|OPTION-L |1 |13|9|S|-        |-|- |-         |- |-|-  | 7|
---------------------------------------------------------------------------------------
| 6|  3|R|2|PRO-R-I    |3M|P|SCREEN-R |- |- |-|S|5-DROP   |8|89|HITCH     |VS|2|COM| 8|
---------------------------------------------------------------------------------------
| 6|  4|R|R|VEER-L     |1L|P|108F     |- |- |-|S|7-DROP   |7|88|FLY       |VL|9|INC| 0|
---------------------------------------------------------------------------------------
| 6|  5|R|3|SPLIT-T    |2L|R|SWEEP-L  |2 |22|8|S|-        |-|- |-         |- |-|   | 4|
---------------------------------------------------------------------------------------
| 6|  9|R|4|PRO-L-I    |1L|P|SCREEN-L |- |- |-|S|5-DROP   |4|44|CURL      |VS|1|COM| 4|
---------------------------------------------------------------------------------------
 AVERAGE = 3.5       MODE =  4    TOTAL NUMBER OF YARDS = 21
 MEDIAN = 4   MEAN = 3.75         NUMBER OF KEYED PLAYS = 6
 THIS IS 13% ALL THE PLAYS
 TOTAL PLAYS ENTERED = 45
```

```
    MATRIX CHART

  HASH BY HOLE CHART
 |  9  |  7  |  5  |  3  |  1  |  0  |  2  |  4  |  6  |  8  |
-------------------------------------------------------------
L|   1 |   0 |   2 |   1 |   0 |   0 |   1 |   0 |   1 |   1 |
 |     |     |     |     |     |     |     |     |     |     |
M|   1 |   0 |   0 |   1 |   1 |   1 |   0 |   1 |   0 |   0 |
 |     |     |     |     |     |     |     |     |     |     |
R|   3 |   0 |   1 |   0 |   1 |   0 |   0 |   1 |   0 |   1 |
-------------------------------------------------------------
     5     0     3     2     2     1     1     2     1     2
    26%   0%    16%   11%   11%   5%    5%    11%   5%    11%
```

You will now be asked for some information about the charts that you want to have created.

```
DO YOU WANT THE CHARTS WRITTEN TO A FILE TO
BE PRINTED OR TO THE TERMINAL?   T/F
```

This is where you decide if you want the charts to be shown on your terminal or to be put into a file so that you can have paper copies printed. If you want them written to a file to be printed on paper type "F" if you want them written to your terminal then type "T". If you have them written to a file remember that the name of the file will be the team name that you used to begin the program followed by a period and the letters "OC" e.g. BOBCATS.OC. You may put as many files as you want in one output file during each run, but remember, that if you turn the program off and start again with the same team name, the old files may get deleted.

Next you will be asked if you want list charts or matrix charts.

```
DO YOU WANT LIST CHARTS OR MATRIX CHARTS?  ENTER
          L for LIST CHARTS
          M for MATRIX CHARTS
```

If you want list type charts you should type "L", if you want matrix charts you should type "M".

If you have chosen list charts you will be asked the following questions about how and what you want the chart to be keyed on.

```
DO YOU WANT THE CHARTS IN GAIN-LOSS ORDER OR
IN THE ORDER THAT THEY WERE ENTERED?
      O for GAIN LOSS ORDERED
      D for AS ENTERED
```

Do you want the charts in ascending order of gain or loss, enter a "O", if you want them in the order they were entered, type a "D".

```
 HOW MANY ITEMS DO YOU WANT THE CHART KEYED ON?
```

At this time you must decide how many items (from 1 to 20) that you want the list charts to be keyed on. You may enter any number from 1 to 20 and you will be asked for the value of that many keys by the questions that follow the menu.

```
WHAT DO YOU WANT THE FIRST KEY TO BE?

    TYPE THE APPROPRIATE NUMBER TO INDICATE THE KEY AREA
 1: DRIVE NO                        11: HOLE
 2: PLAY NUMBER                     12: STRONG OR WEAK SIDE
 3: HASH MARK                       13: RESULT-GAIN OR LOSS
 4: ZONE                            14: PASS ACTION
 5: DOWN and DISTANCE               15: POSITION OF RECEIVER
 6: FORMATION                       16: NUMBER OF RECEIVER
 7: RUN or PASS                     17: PASS RESULT
 8: PLAY                            18: PASS PATTERN
 9: POSITION of BALL CARRIER        19: DEPTH OF PASS
10: NUMBER OF BALL CARRIER          20: ZONE OF PASS
21: ALL PLAYS
```

You should now decide what report(s) you want and enter the correct key number. Next you will get one of the sixteen questions that follows to find out what you want the key values to be:

```
WHICH DRIVE NUMBER DO YOU WANT THE PLAYS FOR?

WHICH PLAY NUMBER DO YOU WANT THE PLAYS FOR?

WHICH HASH MARK DO YOU WANT THE PLAYS FOR?

WHICH YARDAGE ZONE DO YOU WANT THE PLAYS FOR?

WHICH DOWN AND DISTANCE DO YOU WANT THE PLAYS FOR?

WHICH FORMATION DO YOU WANT THE PLAYS FOR?

DO YOU WANT THE RUNNING PLAYS OR THE PASSES?

WHICH PLAY DO YOU WANT THE CHART FOR?

WHICH BALL CARRIERS POSITION DO YOU WANT TO CHECK?

WHICH BALL CARRIER DO YOU WANT TO SEE THE PLAYS FOR?

WHICH HOLE DO YOU WANT THE INFORMATION FOR?

DO YOU WANT THE STRONG OR WEAK SIDE PLAYS?

WHAT GAIN OR LOSS VALUE DO YOU WANT TO SEE THE PLAYS FOR?

WHICH PASS ACTION DO YOU WANT THE PLAYS FOR?
WHICH PASS RESULT DO YOU WANT TO HAVE A CHART FOR?

WHICH RECEIVER POSITION DO YOU WANT A CHART FOR?

WHICH RECEIVER NUMBER DO YOU WANT A CHART FOR?

WHICH PASS PATTERN DO YOU WANT A CHART FOR?

WHAT PASS DEPTH DO YOU WANT A CHART FOR?

WHAT PASS ZONE DO YOU WANT A CHART FOR?
```

You must answer these questions with the exact same thing with which you entered the information with. If you indicated that you wanted more than one key, you will be asked for the next key value. This will continue until you have entered values for each of the keys for which you asked. The chart that you have requested will now be written to a file or to your terminal depending on where you asked for it to be written.

After the program has built the chart that you have requested, you will be given a chance to go back and request another list chart.

```
DO YOU WANT ANOTHER LIST CHART?   Y/N
```

If you want more list charts type "Y" and you will be returned to the list chart part of the program. If you have all the list charts that you want, type "N" and you will be asked if you want any more offensive charts.

```
DO YOU WANT ANY MORE OFFENSIVE CHARTS?  Y/N
```

If you want another chart type "Y" and the program will go back and let you choose another chart. For example you may now want matrix charts or charts ordered by gain, instead of as entered. All of the charts from this run will be put into the same file, so that they may all be printed at once. When you have all of the offensive charts that you want type, "N" and the program will ask you if you are finished or if you want to run it longer.

```
DO YOU WANT ANOTHER RUN?  Y/N
```

If you want to do another run for entering more plays, scouting defense, or getting more charts type "Y" to get back to the start of the program. If you are finished for now, type "N" and the program will stop.

If you requested Matrix charts, you will be asked to select which type of matrix chart you want.

```
ENTER THE NUMBER THAT INDICATES THE TYPE OF
MATRIX CHART THAT YOU WANT.
         1 for HOLE BY DOWN CHART
         2 for HOLE BY HASH CHART
         3 for HOLE BY FORMATION CHART
         4 for PLAY BY DOWN CHART
         5 for PASS ACTION BY HASH CHART
         6 for PASS ROUTE BY HASH CHART
         7 for FORMATION BY HASH CHART
         8 for PASS ACTION BY ZONE CHART
         9 for PASS ROUTE BY ZONE CHART
        10 for PASS ACTION BY DOWN CHART
        11 for PASS ROUTE BY DOWN CHART
        12 for FORMATION BY DOWN CHART
        13 for FORMATION BY PLAY CHART
        14 for ALL THE CHARTS
```

When you enter the number of the type of chart you want, it will be written to a file or to your terminal depending on where you asked for it to be written.

You will now be asked if you want another matrix chart.

```
DO YOU WANT ANOTHER MATRIX CHART?
```

If you want another chart type "Y" and the program will go back and let you choose another chart. All of the charts from this run will be put into the same file so that they may all be printed at once. When you have all of the offensive charts that you want, type "N" and the program will ask you if you are finished or if you want to run it longer.

```
DO YOU WANT ANOTHER RUN?  Y/N
```

If you want to do another run for entering more plays or getting more charts, type "Y", to get back to the start of the program. If you are finished for now, type "N" and the program will stop.

GLOSSARY

BLITZ: A blitz is the action taken by any of the defensive players that are not normally lined up on the line of scrimmage when they go across the line to rush the passer instead of backing up to cover a receiver.

COVERAGE: The type of defense that is used to cover the receivers to try to prevent them from catching a pass. The two main types are man-to-man when each defender has a specific receiver to cover where ever he goes on the field, and zone when each defender has a specific area of the field to cover regardless of which receiver comes into the area. There are many variations on each of these coverages.

DEPTH OF PASS: This is how far beyond the line of scrimmage the football travels in the air between the passer and the receiver. For this program the depths are divided into zones but this is often refered to in exact yardage.

DOWN and DISTANCE: In the game of football a team is given four chances called downs to advance the ball ten yards. Down and distance is the number of the current down and the distance needed to reach the ten yard advance.

DRIVE NUMBER: A drive is a series of plays that a team uses to try to advance the ball to the end zone and score points. During the course of a game a team will have several of these chances and the drive number is a number to indicate to which of these chances we are referring.

FORMATION: An offensive team can position its players in many different places on the field to try to make the play they will run have the best chance of success.

FRONT: This is the number of players and the way they line up that the defensive team has on the line of scrimmage.

GAIN or LOSS: This is the number of yards that the offensive team moves the ball from the line of scrimmage on any play.

HASH MARK: A hash mark is a mark 15 yards from each side of the field that marks the closest place to the sideline that the ball may be spotted to begin a play. Right or left is to be considered to be to the right of left as in respect to the offensive players.

HOLE: In football a hole is an area on the line of scrimmage that the ball carrier runs to to try to advance the ball down the field. Holes a usually designated by numbers with the zero hole being over the center and the odd numbers from 1 to 9 to the left and the even numbers from 2 to 8 to the right.

MOTION: If any of the offensive players are moving, from one position to another before or during the time that the ball is put into play.

NUMBER OF PLAYER: This is the number that each of the players wears on his uniform.

OPPONENT: This is the team against which the team being scouted is playing the game.

PASS: One of the two basic types of plays used in the game of football. A pass is a play in which one player throws the ball forward to another player.

PASS ACTION: This is what the quarterback (or other player who passes the ball) does between the time he gets the ball from the center and the time he actually throws it.

PASS PATTERN: This is the route that the pass receivers take to get to the area where they will catch the ball.

PLAY: In football a play is what ever action the offensive team takes to try to advance the ball toward the goal line. A play is decided on before the start of the action and each player has a designated job to do in each one.

PLAY NUMBER: Each drive consists of a number of plays. The play number is which one of the plays it is.

POSITION OF PLAYER: This is the position in the formation that the player was at the start of the play. This can be designated by a number, a letter, or a descriptive word.

RESULT OF PASS: There are three things that can happen when the ball is passed, the result is which of these happened. The possibilities are complete, incomplete, or intercepted.

RUN: This is a play in which a ball carrier receives the ball either directly from the center or from the quarterback and runs with it to try to gain yardage.

STRONG SIDE: This is the side of the offensive formation that has the most players lined up on the line if scrimmage.

STUNTS: This is any method that the defensive linemen use to stop the passer or ball carrier other than just moving straight across the line of scrimmage.

WEAK SIDE: This is the side of the offensive formation that has the least players lined up on the line if scrimmage.

YARDAGE ZONE: For scouting purposes a football field is broken up into zones. These zones are usually designated as follows:

1. goal line to the 10 yard line.
2. 10 yard line to the 30 yard line.
3. 30 yard line to the 50 yard line.
4. 50 yard line to the 30 yard line.
5. 30 yard line to the 10 yard line.
6. 10 yard line to the goal line.