BNL--45766

DE92 007624

DEVELOPMENT OF POLYMER CONCRETE FOR DIKE INSULATION AT LNG FACILITIES, PHASE IV, LOW COST MATERIALS

FINAL REPORT
SEPTEMBER 1, 1987 — APRIL 30, 1990

Lawrence E. Kukacka

Work Performed by: Jack J. Fontana, Walter Reams, and David Elling

January 1991

Prepared by the
GAS RESEARCH INSTITUTE
ENVIRONMENTAL AND SAFETY RESEARCH DEPARTMENT
CHICAGO, ILLINOIS 60631

ENERGY EFFICIENCY AND CONSERVATION DIVISION
DEPARTMENT OF APPLIED SCIENCE
BROOKHAVEN NATIONAL LABORATORY
ASSOCIATED UNIVERSITIES, INC.

This work was performed under the auspices of the U.S. Department of Energy
Washington, D.C. Under Contract No. De-AC02-76CH00016

PB92-101

REPORT DOCUMENTATION PAGE | 1. REPORT NO. | 2. | 3. Recipient's Accession No. GRI90/0259

4. Title and Subtitle
Development of Polymer Concrete for Dike Insulation at LNG Facilities, Phase IV, Low Cost Materials

5. Report Date
December 1990

6.

7. Author(s)
Lawrence E. Kukacka

8. Performing Organization Rept. No.

9. Performing Organization Name and Address
Energy Efficiency and Conservation Division
Department of Applied Science
Brookhaven National Laboratory
Upton, N.Y. 11973

10. Project/Task/Work Unit No.
BNL-45766

11. Contract(C) or Grant(G) No.
(C) 5084-252-1144
(G)

12. Sponsoring Organization Name and Address
Gas Research Institute
Environmental and Safety Research
Chicago, IL 60631

13. Type of Report & Period Covered
Final September 1987 to April 1990

14.

15. Supplementary Notes

16. Abstract (Limit: 200 words)

Earlier GRI-sponsored work at Brookhaven National Laboratory has resulted in the development and utilization of insulating polymer concrete composites (IPC) as a means of reducing the evaporation rate of liquified natural gas in the event of a spill into a containment dike, thereby improving the safety at these sites. Although all of the required properties can be attained with the IPC, it was estimated that a low-cost replacement for the expensive organic binder would be necessary before use of the material would be cost-effective. In the current program, several latex modified cement formulations were evaluated and the most promising one identified. A mixture of two carboxylated styrene-butadiene latexes was selected for use in detailed laboratory property characterizations and a subsequent field evaluation. When compared to the properties of IPC, the latex-modified insulating materials display somewhat higher thermal conductivities, greater permeability to water, and reduced strength. However, these properties still meet most of the performance criteria, and the unit cost of the material ($0.29/lb) is less than one-fifth that of IPC made with epoxy binders. When installed as a 0.75-in. thick overlay, material costs are estimated to be $1.00/ft^2. Laboratory produced specimens had a density of ~5[illegible] lb/ft^3, compressive strength >1500 psi, tensile strength of >200 psi, flexural strength >400 psi, and a thermal conductivity of <0.20 BTU/hr-ft-°F. Little change in the latter occurs upon immersion in water.

17. Document Analysis a. Descriptors

b. Identifiers/Open-Ended Terms

c. COSATI Field/Group

18. Availability Statement:
Release Unlimited

19. Security Class (This Report)

21. No. of Pages
40

20. Security Class (This Page)
Unclassified

22. Price

(See ANSI-Z39.18) See Instructions on Reverse OPTIONAL FORM 272 (4-77) (Formerly NTIS-35) Department of Commerce

Research Summary

Title	Development of Polymer Concrete for Dike Insulation at LNG Facilities, Phase IV, Low Cost Materials
Contractor	Associated Universities, Inc. Brookhaven National Laboratory GRI Contract No. 5084-252-1144
Principal Investigator	Jack J. Fontana
Report Period	September 1987-April 1990
Objective	The objective of this project was to develop and field evaluate an insulating lightweight concrete composed of low cost materials which can be used to effectively insulate LNG storage tank containment dikes.
Technical Perspective	Cost-effective methods for reducing vapor dispersion distances at storage facilities are needed. Since studies have indicated that the rate and quantity of LNG evaporation are dependent upon the rate of heat transfer from the dike surface to the spilled LNG, insulation of the dikes can substantially reduce boil-off, thereby greatly increasing safety. Earlier GRI-sponsored work at BNL resulted in the development and application of insulating polymer concrete composites. These materials consist of low density multicellular glass and/or ceramic macrospheres bound together with unsaturated polyester or epoxy resins, and they meet all of the necessary property criteria. Unfortunately, it was estimated that a low-cost replacement for the organic binder would be necessary before the materials would be a cost-effective option for reducing dispersion distances. In the current program, latex modified cement mortars were evaluated as a lower cost substitute for the insulating polymer cement.
Results	A series of latex modified cement mortar formulations were evaluated. Based upon these tests, a mixture of two carboxylated styrene-butadiene latexes was selected for use in detailed laboratory evaluations and a subsequent field evaluation. When compared to the properties of the insulating polymer concrete, the latex-modifed materials have somewhat higher thermal conductivities, greater permeabilities to water, and reduced mechanical properties. However, they still exceed most of the performance criteria. Laboratory produced specimens typically had densities ranging between 55 and 60 lb/ft^3, compressive strength >1500 psi, tensile strength >200 psi, flexural strength >400 psi, and a thermal

conductivity <0.20 Btu/hr-ft-°F. The low permeability of the material results in little increase in thermal conductivity upon prolonged immersion in water. Reproducibility of the properties on a larger scale was demonstrated in a field evaluation when the composite was applied to horizontal and vertical surfaces previously insulated with a styrofoam bead concrete. Excellent bonding and low conductivities were achieved, but numerous shinkage cracks were formed, probably due to improper curing. After 9 months in service, no additional deterioration was apparent and the in-situ conductivity was lower than the original values determined from field cast samples. Material costs are estimated to be $0.29/lb or $1.00/ft^2 when applied as a 0.75-in. thick overlay.

Technical Approach

Screening experiments were conducted in which five latexes, produced by four manufacturers, were evaluated. Styrene-butadiene, acrylic and epoxy latexes were included. In conjunction with each of these materials, a variety of insulating type fillers were evaluated. Variables included filler composition and particle size distribution, latex-type, concentration, and the effect of wetting agents. Based upon these tests, a styrene-butadiene-based formulation was used in additional experiments. A detailed property characterization of the mix was made. The formulation was further evaluated in a field test in order to determine if the laboratory-scale test results were reproducible, to establish placement techniques and to determine the long term durability.

Project Implications

The results from the laboratory development and subsequent field evaluation verify that lightweight insulating latex modified cement composites yield properties that make them suitable for use as durable insulation on containment dikes at LNG storage facilities. The cost of latex modified cement is approximately one-fifth that of polyester and epoxy based insulating polymer concretes. This lower cost, plus the added simplicity of installing a portland cement base material, makes latex modified concrete a more cost effective option for insulating LNG impoundment surfaces where high strength as well as good insulating properties are required. As a result, vaporization rates from an accidental spill and the resulting vapor dispersion distances can be greatly reduced at a reasonable cost. Beyond their use at large storage facilities, these materials show considerable promise for hazard mitigation at LNG end-user facilities.

Ted A. Williams
GRI Project Manager

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

INTRODUCTION

Safety at liquefied natural gas (LNG) storage sites has always been of uppermost importance to the natural gas industry. Of primary concern is the accidental spillage of LNG from storage tanks and ancillary piping into earthen containment dikes or those lined with crushed stone.[1,2] When spilled LNG comes into contact with warmer dike surfaces, it vaporizes very rapidly and mixes with the atmosphere to form a hazardous flammable mixture. Depending upon the ambient conditions, these hazardous mixtures can extend downwind for long distances from LNG storage facilities. Analyses of the problem have indicated that the rate and quantity of LNG evaporation are dependent upon the rate of heat transfer from the dike surfaces to the LNG contained in the dike.[3,4] Typically, the maximum evaporation rate occurs within four to eight minutes after a spill. Therefore, since the evaporation rate depends upon the thermal energy transferred from the earth and dike, reductions in the heat flow can result in reductions in the total quantity of LNG evaporated per unit time. One approach to reducing the heat transfer rate is to insulate the dike surfaces, thereby creating a thermal barrier between the walls and floor of the dike and the spilled LNG.

Utilization as a dike insulating material imposes severe requirements. In addition to having low thermal conductivities over temperatures ranging between ambient and -260°F, the insulating material must have a low permeability to insure that the conductivity is not increased due to the absorption of rainwater, be durable under normal weathering conditions, have structural characteristics suitable to support loads from maintenance vehicles, exhibit good bonding to a variety of dike materials, and be cost-effective. Conventional insulating materials do not meet all of these criteria.

In 1983, the Gas Research Institute (GRI) started work at Brookhaven National Laboratory (BNL) to develop materials that met the above criteria. In Phase I of the program which was conducted under GRI Contract No. 5083-252-0812, a lightweight polymer matrix composite which met most of the property criteria was identified.[5] The composite consisted of an unsaturated polyester resin binder and hermetically sealed glass nodules or expanded perlite aggregate. These insulating polymer concrete (IPC) composites have thermal conductivities ranging from 0.08 to 0.15 BTU/hr-ft-°F, water absorptions <2%, low densities (30 to 60 lb/ft^3), and compressive strengths ranging from 1000

to 3000 psi. Two installation methods (precast panels or cast-in-place) appeared to be technically feasible.

In Phase II of the program (GRI Contract No. 5084-252-1144), optimization of the IPC formulation and further property characterization were performed.[6] Attention was focused on improving the shrinkage and fire resistance characteristics of the composite. Evaluations of possible construction techniques were made, and technology for the installation of the IPC on concrete substrates was developed. Cost analyses were also made.

Application methods were further evaluated in Phase III of the program.[7] In this work, it was determined that the IPC formulation could be applied over concrete or crushed stone substrates using shotcreting techniques similar to those used in the concrete industry. Test sections produced by this method exhibited thermal, physical and mechanical properties similar to those for samples made under laboratory conditions. Compared to the use of precast placement methods, significant cost reductions of up to 30% can be accrued by the use of shotcreting. It was also estimated that further reductions in cost would be necessary before the material would be a cost effective option for reducing downwind vapor dispersion distances. These reductions could only be attained by the replacement of the expensive organic binder with lower cost materials such as latex modified portland cement mortars. This was the goal of the Phase IV program, the results from which are described in this report.

PROJECT OBJECTIVE

The objective of this project (Phase IV) was to develop and field evaluate an insulating lightweight latex modified portland cement mortar which can be used effectively to insulate LNG storage tank containment dikes. In addition, as an aid to GRI for the transfer of the technology to the gas industry, a software computer program was to be developed for use in the calculation of LNG boil-off rates from uninsulated and insulated dike surfaces. A video describing the properties and methods for the preparation and placement of the insulating latex modified mortar was also to be prepared.

LABORATORY STUDIES

It is well known that conventional lightweight portland cement concretes have low thermal conductivities when dry.[8] Unfortunately, their open-cell

structures yield large water absorptions resulting in increases in conductivity and decreased weatherability. In an attempt to overcome these deficiencies, the use of latex modified cements in conjunction with closed cell multicellular glass beads was investigated. Since the latex forms a continuous film throughout the portland cement matrix, it was expected to yield a lower permeability mortar or concrete.

A. Materials Selection

A total of five latexes from four different manufacturers were used in initial exploratory experiments. Styrene-butadiene, acrylic and epoxy latexes were included. Descriptions of each are given below.

Type	Identification	Source
carboxylated styrene-butadiene copolymer latex	TYLAC 97-314	Reichhold Chemicals, Inc.
carboxylated styrene-acrylic copolymer latex	SYNTHEMUL DL-8466	Reichhold Chemicals, Inc.
styrene-butadiene polymer emulsion	MOD-A	Dow Chemical U.S.A.
acrylic latex	MC-1834	Rohm and Haas Co.
epoxy emulsion	WDE	Robson-Downes Associates, Inc.

Other materials used in these exploratory experiments are listed below.

Material	Description
Type I Portland Cement	general concrete construction cement
Type III Portland Cement	high early strength cement
Macrolite spheres	multicellular glass spheres with a ceramic coating from 3M company
P2000	free flowing hollow aluminum silicate microspheres from Fillite U.S.A.
52-7-S	hollow aluminum silicate microspheres from Fillite U.S.A.
BYK-A-500	a wetting agent from Byk-Mallinckrodt

The types of hollow spheres used as insulating aggregates in these experiments are shown in Figure 1. Sectioned pieces of the multicellular spheres within a latex-modified cement matrix are shown in Figure 2.

B. Screening Experiments

Based upon the results from Phases I-III of GRI-sponsored research on insulating lightweight composites,[5-7] the following property criteria were established for the purpose of identifying promising materials: compressive strength >1000 psi, water absorption <1%, density 40 to 65 lb/ft^3, and thermal conductivity 0.10 to 0.15 Btu/hr-ft-°F.

The materials described in Section A were used to make a series of lightweight latex-modified mortars for the purpose of determining the workability of the slurries and the density and compressive strength of the cured materials. These results are summarized in Table 1. General conclusions from these tests were as follows. The inclusion of large (No. 4 to 3/8-in. sieve size) macrolite spheres in the slurry (Mix Designs 2 and 8) results in poor workability, and after curing for 7 days, a compressive strength of <1000 psi. The best results were obtained when the particle size of the macrolite spheres ranged between 600 microns and 5.65 mm in diameter. In order to produce a good workable slurry and a cured material with a density in the range of 50 to 54 lb/ft^3, it was necessary to add a small amount of a mixture of free flowing hollow aluminum silicate microspheres having an average particle size of 70 microns and a bulk density of 8 lb/ft^3. This material was identified as P2000 by the supplier Fillite U.S.A. The average compressive strength of these samples at an age of 7 days ranged from 1300 to 1700 psi.

Little effect of the latex composition on the properties of the cured composite was noted. However, two general observations were made; 1) the concentration of the epoxy-based compound (WDE) required was 2 to 4 times greater than those for the other latexes, thereby increasing the cost, and 2) the curing rate for the epoxy was less than those for the other latexes. Based upon these observations, it was decided to eliminate the epoxy latex from further evaluation.

Additional tests were conducted with samples made in accordance with Mix Design 9 in Table 1. Measurements of the compressive strength, flexural strength, and thermal conductivity were made. Estimates of the cost of the material were also made. Each of these results are discussed below.

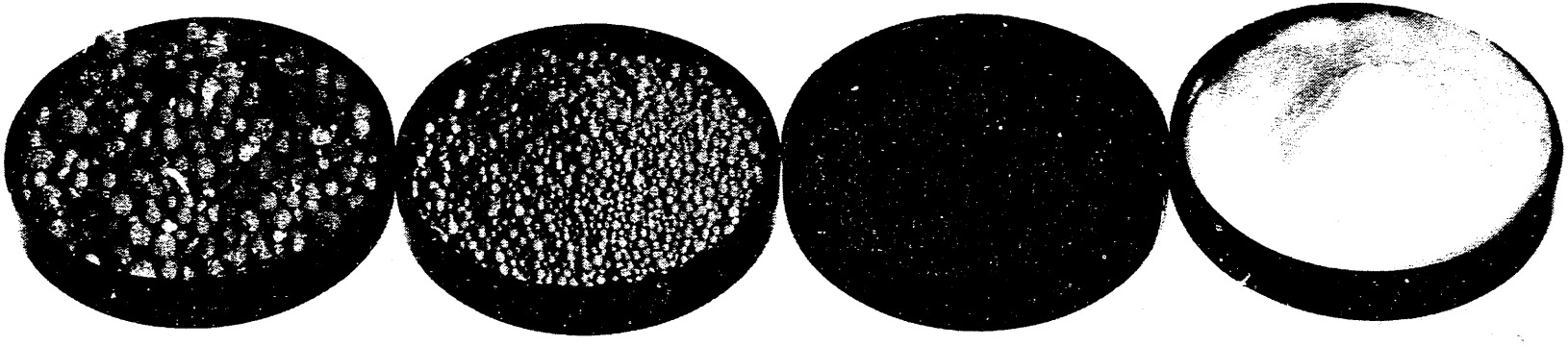


Figure 1. Hollow spheres used as insulating aggregates in lightweight latex-modified concrete composites: from left to right, macrolite spheres sized 3.5/7, 7/14, 14/30 and P2000 powder.

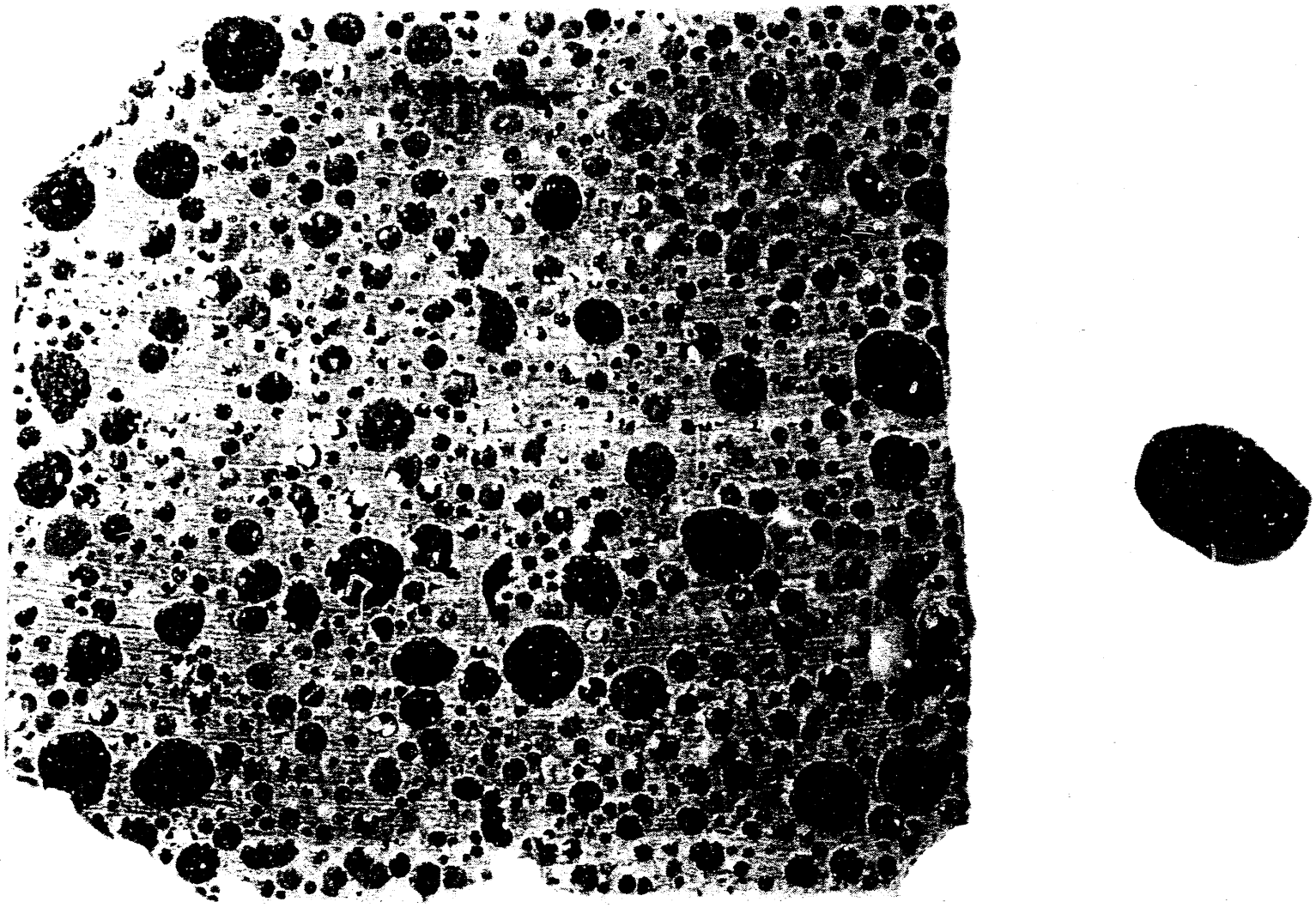


Figure 2. Sectioned pieces of multicellular spheres within a latex-modified cement matrix.

Table 1
Lightweight Latex Modified Cement Insulating Composites, Survey Experiments

Mix Design	Macrolite Spheres[1] 4-3/8 %	3.5/7 %	7/14 %	14/30 %	30/50 %	Fine Filler Type/Quantity %	Cement Type	Cement Quantity %	Latex Type	Latex Quantity %	Water Quantity %	Additives	Density lb/ft^3	Compression Strength psi	Workability
1			11.17	15.08		P2000[2]/ 7.26	I	37.99	97-314[3]	11.73	16.76		52.8	712	good
2	9.09		10.64	13.71		P2000/ 6.60	I	34.54	97-314	10.66	15.24		47.0	969	poor
3		10.0	10.0	10.0		P2000/ 4.3	I	41.6	97-314	9.5	14.3		65.7	2251	good
4		10.19	10.19		10.19	P2000/ 3.09	I	41.97	97-314	9.75	14.63		62.7	1336	good
5		10.22	10.22	10.22		P2000/ 3.10	I	42.11	MOD A[4]	9.66	14.49		66.1	1829	good
6		10.34	10.34	10.34		52-7-S[5]/4.39	I	42.63	97-314	8.78	13.17		69.6	1846	good
7		10.15	10.15	10.15		52-7-S/ 6.15	I	41.85	97-314	8.62	12.92				good
8	10.47	8.64	8.64	8.64		P2000/ 3.66	III	35.60	97-314	9.69	14.66		43.3	646	poor
9		10.03	10.03	10.03		P2000/ 4.26	III	41.34	97-314	9.73	14.59		55.5	1512	very good
10		10.03	10.03	10.03		P2000/ 4.26	III	41.34	97-314	9.73	14.59	BYK-A-500[6] 4 Drops	53.4	1388	very good
11		16.13	8.06	8.06		P2000/ 4.52	III	43.87	97-314	10.32	15.48		50.8	1356	good
12		10.03	10.03	10.03		P2000/ 4.26	III	41.34	DL-8466[7]	9.73	14.59			1183	very good
13		10.03	10.03	10.03		P2000/ 4.26	III	41.34	MC-1834[8]	9.73	14.59		59.7	1428	good
14		10.03	10.03	10.03		P2000/ 4.26	III	41.34	DL-8466	9.73	14.59		46.2	552	very good
15		8.71	8.71	8.71		P2000/ 3.69	III	35.88	WDE[9]	23.75	10.55			2031	poor
16		7.88	7.88	7.88		P2000/ 3.34	III	32.46	WDE	40.57	0			1623	good
17		7.98	7.98	14.0		P2000/ 4.26	III	41.41	97-314	9.74	14.62			2297	very good

Notes:
1. Macrolite spheres - multicellular glass spheres with a ceramic coating from 3M Company.
2. P2000 - free flowing hollow aluminum silicate microsphere from Fillite U.S.A.
3. Tylac 97-314 - carboxylated styrene-butadiene copolymer latex emulsion from Reichold Chemicals, Inc.
4. MOD A - styrene-butadiene latex from Dow Chemical Co.
5. 52-7-S - hollow aluminum silicate microspheres from Fillite U.S.A.
6. BYK-A-500 - wetting agent from Byk-Mallinckrodt.
7. DL-8466 - styrene-acrylic latex from Reichold Chemicals, Inc.
8. MC-1834 - acrylic-latex from Rohm and Haas Co.
9. WDE - epoxy emulsion from Robson-Downes Associates, Inc.

1. Compressive Strength

A series of samples were made to determine the strength developed as a function of curing time. The results from these tests, performed in accordance with ASTM procedure C495, are given in Table 2. Each value listed represents the average of three samples. The data indicate an increase in strength from 1032 psi at an age of 1 day to 1728 psi after 5 days. The strength remained essentially constant thereafter.

2. Flexural Strength

Flexural strength measurements were made using the procedure described in ASTM C78-75. Beams 2-in. x 2-in. x 12-in. long were used in these tests. The data, summarized in Table 3, indicate an average strength of 459 psi. The standard deviation was ±67 psi. The value for the tangent modulus as calculated from the fracture deflection curves was 456,477 psi. The standard deviation was ±22,340 psi.

3. Thermal Conductivity

Preliminary estimates of the thermal conductivity of this mix were also made. Values measured 24 hr after casting ranged from 0.17 to 0.18 BTU/hr-ft-°F. After curing for 4 weeks, the value decreased to the range 0.126 to 0.129 BTU/hr-ft-°F. When the latter specimens were immersed in water for 2 hr and tested 5 minutes after removal from the water bath, they exhibited a thermal conductivity of 0.136 BTU/hr-ft-°F. Compared to the fully cured control, this represents an increase of 5.4%. After exposure to air for 7 days, the thermal conductivity decreased to its original value of 0.129 BTU/hr-ft-°F.

Table 2

Compressive Strength vs Curing Time, Survey Experiments

Cure time, day	Compressive strength, psi[a,b,c,d]
1	1032
3	1624
5	1728
7	1735
9	1674
12	1702
14	1537
28	1834

a, Test procedure, ASTM C495
b, Each value represents average of three samples
c, Specimen size, 3-in. diam x 6-in. long cylinders
d, Latex used, TYLAC 97-314

Table 3

Flexural Strength Results, Survey Experiments

Sample No.[c]	Flexure strength, psi[a,b]	Tangent modulus, psi
49	397	478,406
50	473	441,607
51	417	433,273
52	547	472,503
Average	459	456,477
Standard deviation	±67	±22,340

a, Test procedure, ASTM C78-75
b, Specimen size, 2-in. x 2-in. x 12-in. long beams
c, Latex used, TYLAC 97-314

4. Material Cost Estimate

A preliminary estimate of the cost of the materials for the most promising mix design identified in the survey experiments (Mix Design 9, Table 1), was made. These results, summarized in Table 4, indicate a material cost of $0.29/lb, less than one-fifth the cost of IPC made with epoxy binders.[7] Based upon a density of 55 lb/ft^3 for the lightweight latex modified cement composite and an overlay thickness of 1-in. to provide uniform insulation, this material cost corresponds to $1.33/ft^2 of insulated surface. Due to the similarity of the latex-modified concrete with conventional portland cement concretes, placement costs can be assumed to be equivalent.

Table 4

Material Cost Estimate, Survey Experiments

Material	Quantity, wt %	Material cost, $/lb	Unit cost, $/lb
Macrolite spheres	30	0.50	0.15
P2000	4.5	0.65	0.03
Type III portland cement	41.0	0.10	0.04
Latex	12.0	0.60	0.07
Water	12.5	0	0
			$0.29
		For 0.75-in. thickness,	$1.00/ft^2

C. Characterization Tests

Based upon the results described above, a mix design was selected for use in a series of tests to fully characterize the material prior to using it in a small-scale field evaluation. The composition of this mix is given in Table 5. Three latexes used in the survey experiments (TYLAC 97-314, MOD-A, and MC-1834) were selected for continued characterization. In addition, another latex (TYLAC 68-009) was also evaluated. This carboxylated styrene-butadiene copolymer latex was supplied by Reichhold Chemicals, Inc. and is chemically similar to the TYLAC 97-314 except that it is less odorous. This makes it easier to use in enclosed areas.

Table 5
Mix Design Used in Characterization Experiments

Material	Concentration, % wt
Latex[1]	12.0
Macrolite spheres[2]	
3.5 to 7	10.0
7 to 14	10.0
14 to 30	10.0
Macrospheres Q-Cel 400[3]	4.5
Type III portland cement	41.0
Water	12.5

1) TYLAC 97-314, TYLAC 68-009, MOD-A, and MC-1834 latexes were used. The TYLAC 68-009 is a modification of the 97-314 that is less odorous and designed for use in enclosed areas.
2) Sphere size range in U.S. Mesh
3) Average particle size 75 microns. Supplied by the PQ Corporation.

Properties measured include compressive strength, tensile splitting strength, flexural strength, bond strength, thermal conductivity, and water absorption. The results from these tests are given below.

1. Compressive Strength

Series of specimens consisting of the inorganic constituents previously listed and three carboxylated styrene-butadiene latexes were made for use in compressive strength measurements. These tests were performed in accordance with ASTM procedure C495 using 3-in.-diam x 6-in. long cylindrical specimens.

Compressive strength data for two latexes as a function of curing age are summarized in Table 6. Specimens containing the TYLAC 97-314 latex exhibited a strength of 1032 psi at an age of 1 day. Between ages of 3 and 28 days the strengths were relatively constant and averaged 1690 psi.

The use of the TYLAC 68-009 latex yielded a higher ultimate strength. At an age of 1 day, the strength was 994 psi. This value increased with curing

time up to 4 days where it leveled off at an average value of 2077 psi. Both latexes reached the 1000 psi compressive strength criterion within 24 hr and had an ultimate strength far above it.

Table 6

Compressive Strength vs Curing Time, Characterization Tests

Cure time, day	Compressive strength, psi[a,b,c] Latex type TYLAC 97-314	TYLAC 68-009
1	1032	994
2	--	1351
3	1624	--
4	--	1964
5	1728	1738
6	--	1999
7	1735	2035
8	1674	--
9	1702	2022
12	1537	2274
14	--	2832
28	1834	2203

a, Test procedure, ASTM C495
b, Each value represents average of three samples
c, Specimen size, 3-in.-diam x 6-in.-long cylinders

The TYLAC 68-009 latex was also used in a series of tests to determine the effects of temperature on the strength of the composite. Measurements were made at -50°, 70° and 140°F. As shown in Table 7, the strength decreased with increased temperature from an average value of 3060 psi at -50°F to 1445 psi at 140°F. Similar trends were noted for previously developed IPC systems.[6]

For comparative purposes, specimens prepared with the MC-1834 were also tested. The formulation used for these specimens is identified in Table 1 as Mix Design No. 13. The results from these tests, also given in Table 7, indicate trends similar to those obtained with the styrene-butadiene latex (TALAC 68-009). Over the temperature range -50° to 140°F the compressive strength decreased 48% from an average value of 2865 psi to 1498 psi. The styrene-butadiene latex-based specimens decreased 53%. However, at 140°F, both materials exceeded the design criterion and are strong enough to support maintenance vehicles or other normal load requirements.

2. Tensile Splitting Strength

The tensile splitting strength as a function of temperature was measured for a series of 3-in. diam by 6-in. long specimens containing the TYLAC 68-009 latex. The test procedure was in accordance with ASTM C496-71. Trends similar to those for the compressive strength were obtained. At -50°F the average tensile splitting strength was 484 psi. This decreased to 228 and 210 psi at 70° and 140°F, respectively. These data are given in Table 8.

Similar trends were obtained for specimens containing the acrylic latex MC-1834, but the extent of strength regression with increasing temperature was less. The strength decreased 51% from a value of 441 psi at -50°F to 214 psi at 140°F. Over the same range of temperature, specimens containing the TYLAC 68-009 exhibited a 56% reduction.

3. Flexural Strength

Flexural strength tests were performed on samples containing the three polymer latexes. The tests were performed in accordance with ASTM procedure C78-75 using 2-in. x 2-in. x 12-in. long beam samples. The results are summarized in Table 9, and they indicate similar strengths (~425 psi) for the TYLAC 68-009 and MOD-A-based formulations. The MC-1834 acrylic latex formulation yielded a strength of 284 psi, a value ~33% lower.

Table 7

Compressive Strength at Various Temperatures

Latex type	Compressive strength, psi, at test temperature, °F		
	-50°	70°	140°
TALAC 68-009	3190	2472	1588
	3121	2564	1580
	2871	2061	1168
AV.	3060	2366	1445
MC-1834	2810	2149	1593
	3347	1974	1536
	2438	1978	1365
AV.	2865	2034	1498

Test procedure, ASTM C495
Specimen size, 3-in.-diam x 6-in.-long cylinders

Table 8

Tensile Splitting Strength at Various Temperatures

Latex type	Tensile splitting strength, psi, at test temperature, °F		
	-50°	70°	140°
TALAC 68-009	491	238	233
	472	228	199
	488	218	200
AV.	484	228	210
HC-1834	488	308	231
	407	243	202
	428	300	209
AV.	441	284	214

Test procedure, ASTM C496-71
Specimen size, 3-in.-diam x 6-in.-long cylinders

4. Bond Strength to Concrete Substrates

The bond strength in tension of a lightweight latex modified mortar containing the TYLAC 68-009 latex was determined using the method described in American Concrete Institute (ACI) Standard 503R-80. Two types of concrete substrates were considered. The first was a normal density portland cement concrete and the other was a lightweight concrete containing styrofoam beads, sometimes referred to as expanded polystyrene concrete (EPS). The latter is of interest since it has been installed at some LNG facilities as an insulating material, and the ability to bond to it will be of great importance during retrofit operations.

Slabs, 4 ft by 4 ft by ~4-in. thick, of each material were cast and allowed to fully cure. After cleaning the top surface by sandblasting, a primer coat consisting of 50% latex (TYLAC 68-009) and 50% portland cement was brushed on immediately prior to placing a 0.75-in. thick lightweight insulating overlay. A typical section from a portland cement concrete slab insulated with a latex-modified concrete overlay is shown in Figure 3. In the case of the bond test with the conventional concrete substrate, failure always occurred in the insulating overlay. The average tensile bond strength was 250 psi, in good agreement with the previously discussed tensile strength value of 228 psi. With the lightweight concrete slabs containing the expanded polystyrene beads, failure was always in the concrete substrate. The average tensile bond strength was a very low 106 psi and it is indicative of the poor strength and durability characteristics of this type of concrete. The ability to bond latex modified insulating lightweight concrete overlays to other insulating-type concretes should not be a constraint for possible remedial field applications.

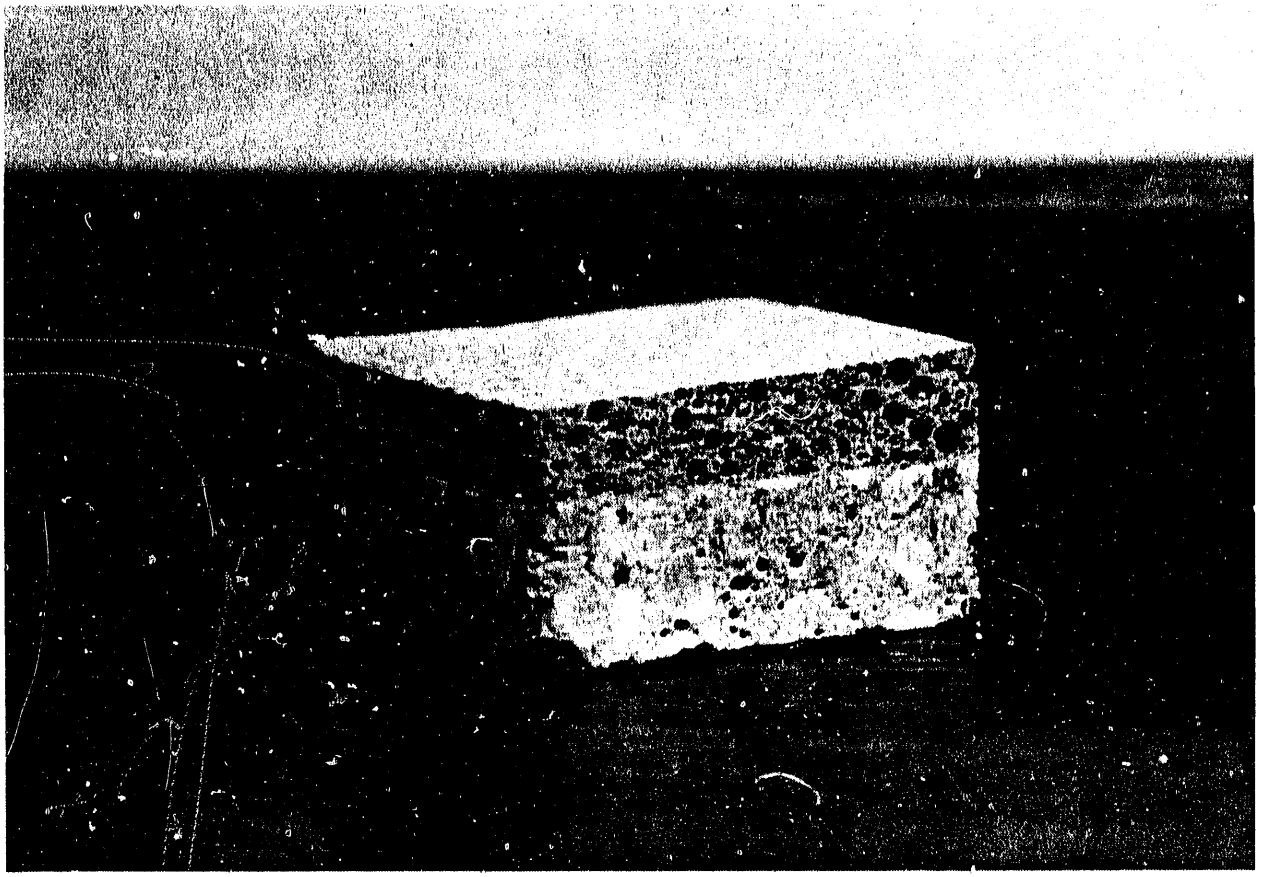


Figure 3. Typical section from a conventional portland cement concrete slab (bottom) insulated with a latex-modified concrete overlay (top).

Table 9

Flexural Strengths and Modulus of Lightweight Insulating Composites

Latex type	Flexural strength, psi	Flexural modulus, 10^5 psi
TYLAC 68-009	545	4.69
	378	6.95
	349	6.04
AV.	424	5.89
MC-1834	294	6.47
	277	6.85
	281	5.60
AV.	284	6.31
MOD-A	425	3.83
	511	5.60
	349	5.34
AV.	428	4.92

Test procedure, ASTM C78-75
Specimen size, 2-in. x 2-in. x 12-in. beams

The effect of thermal shock on the bond was also evaluated. In these tests, liquid nitrogen was poured onto the surfaces of insulated slabs prepared in the manner described above. After two thermal cycles, neither cracking or disbondent of the insulating overlay were apparent. Upon applying a tensile load, bond strength results similar to those reported above were obtained. Namely, failure of the slab containing the EPS concrete substrate occurred within the substrate, and with the conventional concrete substrate, the failure was in the overlay.

5. Thermal Conductivity

A series of plaques 7 by 7 by 1-in. thick were made for use in measurements of the thermal conductivity after exposure to air and water. A Shotherm QTM-D2 Quick Thermal Conductivity Meter, manufactured by Showa Denko K.K., was used. The samples tested were composed in accordance with the optimized composite mix design and contained TYLAC 97-314, TYLAC 68-009, MOD-A and MC-1834

latexes. These results are given in Table 10. With the TYLAC-type latexes, the thermal conductivity decreased during the first 7 days as hydration of the portland cement progressed After curing in air for 7 days, the values ranged between 0.131 and 0.136 BTU/hr-ft-°F. The specimens were then fully immersed in water for 24 hr. Measurements made immediately upon removal of the plaques from the water indicated conductivities essentially the same as those when the specimens were first cast (0.19 to 0.21 BTU/hr-ft-°F). However, within 24 to 48 hr of exposure to a laboratory air environment, the conductivities decreased to their pre-immersion values.

Tests were also performed on specimens containing MOD-A and MC-1834 latexes. Results obtained were similar to those from the TYLAC-containing samples.

The effect of prolonged immersion in water on the thermal conductivity was also investigated. In this work, 7-in. x 7-in. x 1-in. thick plaques, made in accordance with the formulation listed in Table 10, were soaked in water at ambient temperature for 11 days. The test results, summarized in Table 11, indicate similar trends for each of the latex systems. When compared to the conductivities of air dried samples, all three latex systems exhibited significant increases within 24 hr, after which they remained relatively constant for the 11 day period. Samples containing the MOD-A and MC-1834 latexes had thermal conductivities of 0.194 and 0.195 BTU/hr-ft-°F, respectively, upon saturation with water. These values represent increases when compared to the controls of 19 and 26%, respectively. The TYLAC 68-009 sample increased approximately 32%, thereby indicating a greater amount of porosity accessable to water. This is discussed further in Section 6, Water Absorption.

Table 10

Thermal Conductivity Results

Cure time, day	Latex type: TYLAC 97-314	TYLAC 68-009	MOD-A	MC-1834
	Thermal conductivity, BTU/hr-ft-°F			
1	0.193	0.210	0.180	0.182
2	--	0.184	--	0.163
3	--	0.181	--	--
4	0.137	--	--	--
5	0.136	--	--	--
6	--	--	--	--
7	--	0.131	--	0.156

Sample Composition		
Macrospheres	3.5 - 7	10.0%
	7 - 14	10.0%
	14 - 30	10.0%
Macrospheres Q-Cel 400		4.5%
Type III Cement		41.0%
Water		12.5%
Latex		12.0%

Table 11

The Effect of Water Immersion on Thermal Conductivity

Immersion time, day	Thermal Conductivity, BTU/hr-ft-°F Latex type: TYLAC 68-009	MOD-A	MC-1834
0	0.171[a]	0.165[a]	0.156[a]
1	0.227	0.197	0.197
2	0.231	0.190	0.197
3	0.225	0.192	0.197
4	0.224	0.193	0.193
7	0.220	0.193	0.188
8	0.220	0.193	0.192
9	0.219	0.193	0.192
10	0.216	0.197	0.198
11	0.216	0.202	0.200

a, Air dried specimens
Sample size, 7-in. x 7-in. x 1-in. thick

6. Water Absorption

The water absorption of polymer modified lightweight concrete was measured using two different techniques. The first involved determinations of the change in weight after saturation in water. An electrical resistivity method as described in AASTHO-T277-81 was the second method used.

Test results from weight change measurements are summarized in Table 12. In these tests, samples of two sizes, 3-in.-diam x 6-in.-long cylinders and 7-in. x 7-in. x 1-in. thick plaques, were evaluated after immersion in water at ambient temperature for times up to 11 days. Three polymer latexes at a concentration of 12 wt% were compared. The following generalizations are apparent: 1) cylindrical-shaped specimens had significantly lower water absorptions than the plaques, probably due to their lower surface/volume ratio and the relative ease of compaction during casting, 2) the use of the MOD-A styrene-butadiene polymer emulsion resulted in the lowest water absorptions, and 3) the data for the TYLAC 68-009 and MC-1834-containing specimens exhibit a great deal of scatter, but in general plaque-shaped samples containing the

Table 12

Water Absorption Results

Immersion time, day	Latex type: Sample size	Water absorption, wt% TYLAC 68-009 C	TYLAC 68-009 P	MOD-A C	MOD-A P	MC-1834 C	MC-1834 P
1		2.92	6.3	0.36	4.0	2.37	8.2
2		3.08	6.5	0.50	4.0	2.63	8.5
3		3.20	6.5	0.58	4.3	2.80	8.6
4		3.29	6.6	0.70	4.4	3.04	8.8
7		3.57	6.9	0.96	4.7	3.52	9.0
8		3.65	6.9	1.00	4.8	3.55	9.0
9		3.70	6.9	1.06	4.8	3.62	9.0
10		3.77	7.0	1.13	4.9	3.74	9.1
11		3.79	7.0	1.15	4.9	3.77	9.1

C, Specimen size, 3-in.-diam x 6-in.-long cylinders
P, Specimen size, 7-in. x 7-in. x 1-in. thick plaques

former had lower water absorptions. Both series of cylindrical-shaped samples had similar values. Comparison of these observations with the thermal conductivity values given in Table 11 for the same specimens fail to reveal a clear relationship between thermal conductivity and water absorption. Based upon theory, it would be expected that the thermal conductivity of the MOD-A specimens would be the lowest, followed by the TYLAC 68-009 and then the MC-1834. This was not observed experimentally where the MOD-A and MC-1834 systems produced samples that had conductivities when water saturated of 0.194 and 0.195 BTU/hr-ft-°F, respectively, and the TYLAC 68-009 system 0.222 BTU/hr-ft-°F. The limited number of specimens tested, and physico-chemical factors affecting the workability of the mix formulations, polymer distribution within the samples, and the bonding to the fillers, may be the cause of these descrepancies.

Additional tests were performed to determine if reductions in the water absorption, and thereby lower thermal conductivities, could be accrued by increasing the polymer content. Two TYLAC 68-009 latex concentrations (12% wt

and 18% wt) were used in these tests which were performed in accordance with AASTHO-T277-81.

The test results indicated a permeability value of 991 coulombs for the sample containing 18% wt latex and 2800 coulombs for the sample with 12% wt latex. Normally, the permeability values for conventional latex-modified mortars (3:1 sand-portland cement) vary from 800 to 2000 coulombs, depending upon the latex concentration and length of cure. Since the water permeability values for the 12% wt latex-containing samples were higher than desired, it was decided that TYLAC 68-010 latex would be used as a partial replacement for the TYLAC 68-009 in a planned small field evaluation. These resins are identical except that the former contains a defoam or antifoam system. It's use was expected to reduce the number of air voids resulting from entrapped air during mixing of the lightweight concrete formulation, thereby reducing the permeability to water.

A series of specimens containing 15% wt TYLAC 68-010 were prepared and the thermal conductivities measured as a function of curing time in air. These data are summarized in Table 13. The values decreased from

Table 13

Thermal Conductivity Results for Latex Containing Antifoam Agents

Cure time, day	Thermal conductivity, [a,b] BTU/hr-ft-°F
1	0.173
4	0.165
5	0.160
6	0.156
7	0.158
11	0.164
15	0.153
25	0.152
32	0.155

a, Latex, 15% wt TYLAC 68-010
b, Specimen size, 7-in. x 7-in. x 1-in. thick plaques

0.173 BTU/hr-ft-°F at an age of 1 day to 0.156 BTU/hr-ft-°F after 6 days. Beyond that age, little further reduction occurred. A comparison of these results with those obtained earlier with samples containing 12% wt TYLAC 68-009 (Table 10), indicates that at early ages the TYLAC 68-010 formulation had lower thermal conductivities (0.165 vs 0.181 BTU/hr-ft-°F). However, at ages beyond 7 days the TYLAC 68-009 had lower values (0.131 vs 0.158 BTU/hr-ft-°F). The reason for this may be that the water content in the former was decreased from 12.5 wt% to 9.5 wt% in order to accommodate the increased latex concentration. This reduction in initial water content would be expected to result in a lower thermal conductivity product at early curing ages. However, upon further hydration of the cement, the increased density of the cured concrete would be expected to yield a less porous and therefore higher conductivity product. Also, upon immersion in water, the reduced porosity would be expected to maintain the conductivity closer to its air dried value. Unfortunately, this was not verified experimentally. Based upon these data, a 50:50 mixture of the two latexes was selected for use in the field evaluation. This ratio represented a compromise between product density and porosity as they affect thermal conductivity in wet environments. Time and budgetary constraints prevented a quantitative evaluation of these parameters.

FIELD EVALUATION

In order to determine if the results obtained in the laboratory characterization of lightweight latex modified composites could be reproduced in the field and to demonstrate possible placement techniques, a small field evaluation was conducted. Plans were made and later implemented to place an insulating overlay on approximately 1650 ft^2 of horizontal surface and 150 ft^2 of verticle surface at a concrete-lined LNG gathering sump, and to compare the results with those from a polystyrene foam lightweight concrete which had been commercially installed earlier to insulate the sump. Some of this insulation had subsequently been treated with an epoxy sealer as a repair technique. Details of this work are given below.

A. Characterization of Existing Lightweight Concrete

Measurements were made to determine the thermal conductivity of the expanded polystyrene (EPS) lightweight cement concrete insulation that was in service at the field test site. For use in these tests, several small pieces of the EPS were removed and sent to BNL. The samples were shipped and stored

in plastic bags in order to maintain the moisture content at a level approximating that when in service. The thermal conductivities of these samples which contained between 25 and 35 wt% water, ranged from 0.35 to 0.50 BTU/hr-ft-°F. After oven drying the samples, this value decreased to the range 0.08 to 0.10 BTU/hr-ft-°F. In contrast, samples of the BNL developed lightweight latex modified insulating concrete had a thermal conductivity of ~0.20 BTU/hr-ft-°F immediately following full immersion in water for 11 days and a pre-immersion value of ~0.13 BTU/hr-ft-°F within 48 hr upon exposure to laboratory air conditions.

Since the field evaluation would necessitate placement of the latex modified insulating concrete on the EPS concrete, attempts were made to measure the bond strength between these materials. In these tests, the surface of the EPS concrete was sandblasted prior to application of the latex modified insulation. Visual inspection of the interface indicated good bonding, but unfortunately the small sample size precluded the performance of tensile bond measurements. Based upon this observation and the results from earlier laboratory studies in which the bond strength to an EPS concrete was measured (see Laboratory Studies Section C-4), it was concluded that bonding of the two materials could probably be attained in the field.

B. Field Installation

In November 1988 a styrene-butadiene latex modified concrete insulating composite was placed on the sump floor and ~70 ft^2 of side walls at a field site. Descriptions of the materials used and method of placement are given below.

1. Mix Design

The mix design selected for use in the field evaluation is given in Table 14. It should be noted that in comparison to the mix design used for the laboratory characterization work (see Table 5), the latex concentration was increased from 12 to 15% wt. This was done in order to reduce the permeability of the cement matrix, thereby maintaining a low thermal conductivity when exposed to wet environments.

In order to minimize the weighing and mixing of the various constituents at the job site, all of the solid components were mixed and then packaged at BNL in cardboard boxes. Each box contained aggregate sufficient to produce a 2 ft^3 mix.

Table 14

Mix Design Used in Field Evaluation

Material	Concentration, % Wt
Latex[a]	15.0
Macrolite spheres[b]	
3.5 to 7	10.0
7 to 14	10.0
14 to 30	10.0
Grefco HP 520[c]	4.5
Type III portland cement	41.0
Water	9.5

a, Two carboxylated styrene-butadiene latexes, TYLAC 68-009 and TYLAC 68-010, both supplied by Reichhold Chemical, Inc., were mixed in equal quantities.

b, Multicellular glass spheres with a ceramic coating supplied by the 3M Company. Size distribution in U.S. Mesh.

c, Filler supplied by Grefco, Inc. Average particle size 70 microns.

2. Surface Preparation

All of the surfaces that were to be overlaid were scraped and sandblasted. This was followed by additional cleaning which consisted of scraping off as much of the epoxy coating and pieces of delaminated portland cement mortar from the EPS concrete as possible. Both materials had been applied earlier in attempts to repair sections of the EPS concrete. Figure 4 illustrates the condition of the floor surface after the cleaning was completed, and it is readily apparent that considerable variation in the substrate existed. A typical wall section is shown in Figure 5.

3. Installation of Screed Rails

Screed rails were attached to the sump floor in order to facilitate compaction and leveling of the insulating material. Wooden strips 2-in. wide by 1-in. thick and wrapped with Mylar tape were used for the rails. Attachment to the concrete floor was accomplished using 3-in. long screw nails. The

Figure 4. Condition of sump floor prior to placement of latex-modified lightweight concrete insulation.

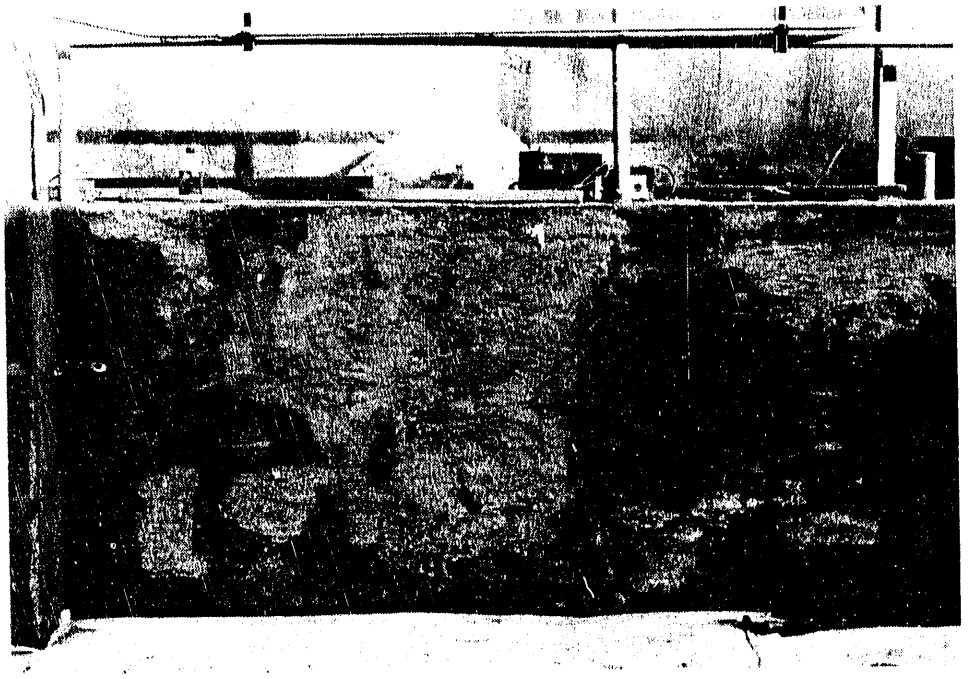


Figure 5. Typical wall section prior to placement of insulating overlay.

rails were installed so as to divide the sump into 6 strips, the four inner ones being ~5-ft wide and the outer ones ~2.5 ft.

4. Mixing, Placement and Finishing

An 8 ft^3 cement mortar mixer was used for the mixing of the insulating latex modified portland cement overlay. First, an amount of prepackaged aggregate sufficient to produce a 2 ft^3 batch was placed in the mixer and mixed for 2 to 3 minutes. The latex and water were then added and all were mixed for 3 to 5 minutes.

The horizontal area to be overlaid was wetted with water and then a slurry containing ~50 wt% latex - 50 wt% cement was broomed onto the surface as a bond coat. The insulating composite mix was transported by wheelbarrow and spread out on the floor in front of the screed using shovels and rakes. A vibratory screed riding on the screed rails was used to level and compact the overlay. Some additional hand finishing was done using magnesium floats.

Several minutes after the overlay was placed, it was covered with a vapor barrier to prevent the evaporation of water from the insulating concrete. In this installation, Transguard 100 (a fibrous mat attached to a polyethylene film) was prewetted and then rolled out onto the surface. The covering remained in place for 24 hr.

The nominal 1-in. thick overlay was placed in strips as indicated in Figure 6. Strips 1 and 5 were ~2.5-ft wide, and strips 2-4 and 6 were 5-ft. Strips 1, 2 and 3 were placed on the first day, Strips 4, 5 and 6 on the second day. A completed section of the floor is shown in Figure 7.

A 17.5-ft long by 4-ft high section of a vertical wall in the sump was also overlaid with the lightweight latex modified mortar. The approximate thickness of the overlay was to be 2-in. The overlay was placed by filling a 2-in. wide annulus between a polyethylene sheet-covered plywood form and the EPS concrete covered wall surface. Compaction of the composite during placement was accomplished using a vibratory screed motor which was mounted on vertical beams attached to the form. Unfortunately, when the overlay was about 50% complete, the form work started to pull away from its anchors. It was possible to re-anchor the form before it collapsed, however, the overlay

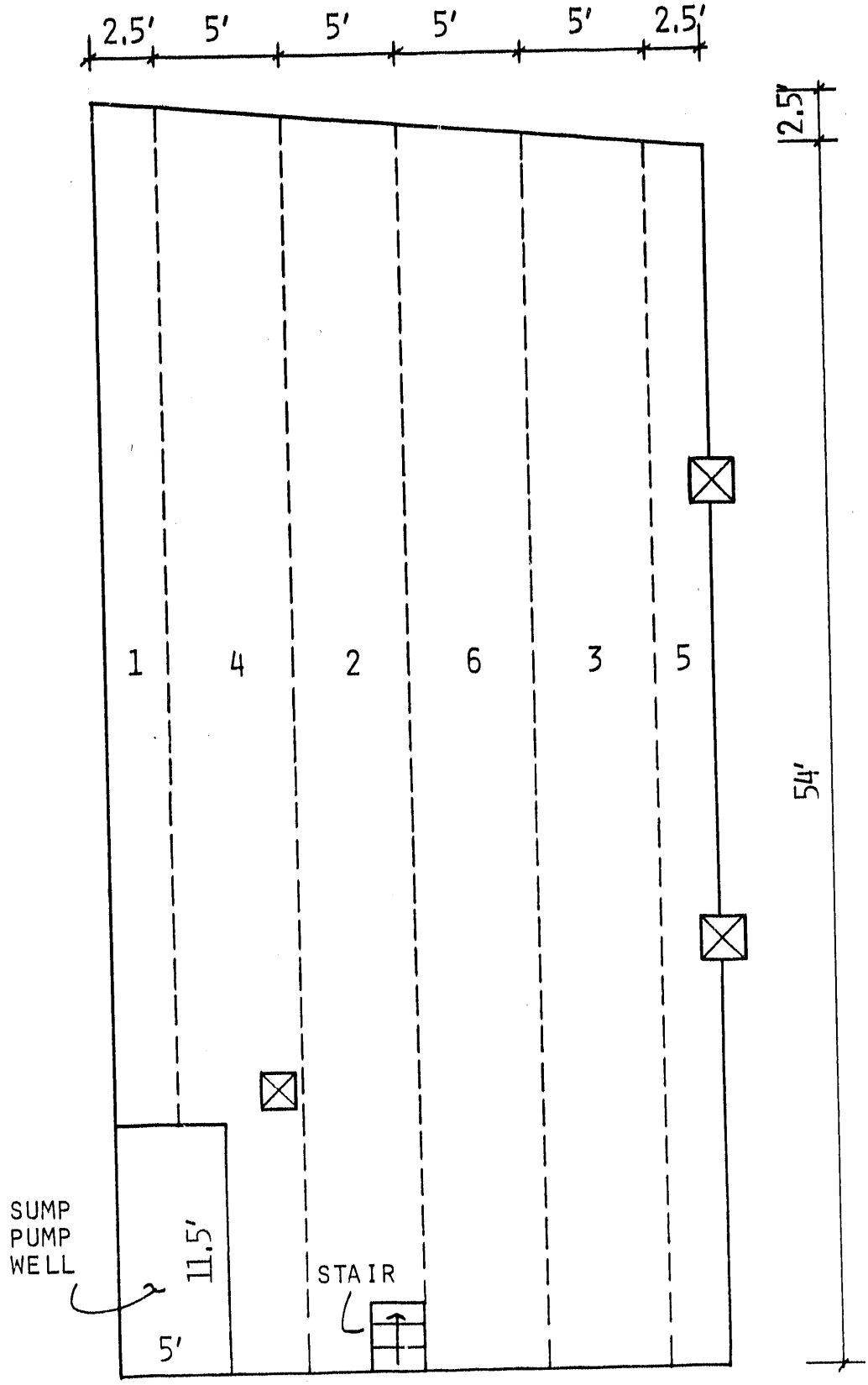


Figure 6. Layout of sump floor.

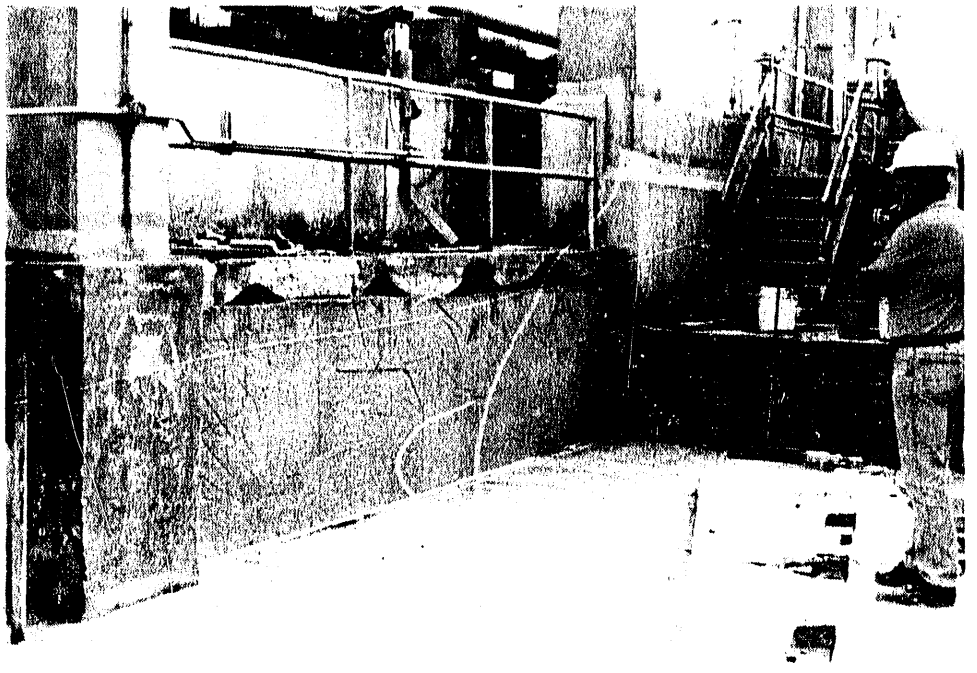


Figure 7. Section of sump floor and wall after placement of insulating overlay.

thickness became irregular varying between 2 and 3-in. A completed section of the wall is shown in Figure 7.

The quantities of material needed for each floor strip and the wall are summarized in Table 15. The placement times required for each section are also given in this Table.

Table 15

Material Requirements for Field Evaluation

Strip No.	Placement time, min	Material required, lb
1	45	600
2	45	1500
3	45	1500
4	40	1800
5	40	800
6	45	1800
wall	60	1100
	Total	9100

5. Materials Cost and Manpower Requirements

As summarized in Table 15, a total of 9100 lb of the latex modified lightweight concrete was placed over floor and wall areas of approximately 1324 ft^2 and 70 ft^2, respectively. Due to the extreme irregularity of the thickness, it is not possible to calculate the material cost on an area basis for the wall. However, for the floor where the overlay thickness was a nominal 1-in., the materials cost of \$0.29/lb (see Table 4) translates to \$1.75/ft^2. This can be compared with the estimated \$1.33/ft^2 that was discussed in Section B-4. The latter was based upon a product density of 55 lb/ft^3. When the field derived density of 64.8 lb/ft^3 is factored into the estimate, this increases the laboratory work-predicted cost to \$1.56/ft^2. The remaining difference can be attributed primarily to variations in the overlay thickness, and secondarily to material losses during mixing and placement.

Due to salary scale differences, the labor costs of BNL Professional and Technical Staff personnel have not been used to calculate placement costs. Time requirements were determined and these are listed below in Table 16. A total of 129 man hr were required to pre-weight the solid constituents in the laboratory, prepare the surface, fabricate and install forms and guiderails, and to mix, place and screed the overlay. It should be noted that the poor initial condition of the concrete substrate resulted in a large labor requirement for surface preparation. This should not be considered typical. Pre-batching of the aggregate represented 20% of the total labor. For a large-scale field installation, a considerable reduction in the time required for this operation should be attainable. The use of larger-scale continuous concrete batching and mixing equipment would also significantly reduce the placement time.

6. Mechanical and Physical Properties Attained

Samples were cast during the placement of the overlay for use in measurements of the mechanical and physical properties of the composite. Compared to earlier specimens prepared in the laboratory in which compaction was performed by use of a vibrating table, the field samples were compacted by hand tamping the molds against the ground or other firm surfaces. The test results from these specimens are summarized in Table 17.

Compressive strength tests were made on 29 cylinders. The average strength was 1788 psi with a coefficient of variation of 24.2%. The average flexural strength of 7 beams was 252 psi and the coefficient of variation was 31.4%. These large variations can be attributed to slight variations in the fluid concentrations in each of the 2 ft^3 batches, and to differences in the degree of compaction of the samples. Earlier laboratory samples had an average compressive strength of 2345 psi and a flexural strength of 424 psi.

Water absorption measurements conducted on 7 samples yielded a value of 0.56%, lower then the samples prepared in the laboratory. The density of 32

Table 16

Field Placement Labor Requirements

Operation	No. of Men	Total time,[a] man-hr
Surface preparation	2	32
Mixing of dried aggregate	2	25
Preparation and placement of forms and screed guides	2	16
Mixing and placement of overlay[b]	4	56
		129

a, Treated area, 1394 ft^2
b, Quantity mixed, 9100 lb

TABLE 17

Mechanical and Physical Properties of Lightweight Latex Modified Mortar Used in Field Evaluation

Sample No.	Compressive strength, psi	Flexural strength, psi	Thermal conductivity, Btu/hr-ft-°F	Density, lb/ft³	Water absorption, %
260		317	0.206	63.4	0.5
261		236	0.191	65.5	0.6
262		307	0.193	62.6	0.7
263		360	0.191	63.2	0.7
264		152	0.193	63.9	0.5
265		226	0.213	61.1	0.45
266		163	0.219	62.7	0.48
267	1171			65.8	
268	1458			69.0	
269	2012			66.3	
270	2149			66.1	
271	1943			66.3	
272	1905			66.4	
273	1088			69.1	
274	2233			69.9	
275	2084			66.1	
276	2000			66.7	
277	1741			65.5	
278	2153			66.6	
279	2783			68.8	
280	1871			64.5	
281	1214			63.2	
282	1099			63.2	
283	1703			63.6	
284	1714			64.5	
285	1970			63.8	
286	1882			63.8	
287	1451			68.8	
288	1718			64.2	
289	1172			60.7	
290	1600			62.7	
291	1313			60.9	
292	2584				
293	2058				
294	1985				
295	1145				
Mean	1788	252	0.200	64.8	0.56
Std. Dev.	433	79	0.011	2.4	0.10
Coef. of Var.	24.2%	31.4%	5.8%	3.7%	18.8%

field-produced samples was 64.8 lb/ft^3, approximately 15% greater than the laboratory values. Lesser agreement was obtained with the thermal conductivity data. In this case the field samples had a thermal conductivity of 0.200 BTU/hr-ft-°F compared to the 0.159 to 0.131 BTU/hr-ft-°F range measured on laboratory samples.

7. Video Documentation

In order to aid in the transfer of the technology developed in this program to the gas industry, video personnel from BNL taped all of the operations performed during the field test. After editing, this tape was combined with others produced in the laboratory to serve as the basis for an instructional video. This video was delivered to GRI.

8. Post-Test Inspection

An inspection of the lightweight latex-modified mortar overlay was made in August 1989, approximately 9 months after installation. The overlay displayed many cracks in a spiderweb pattern but no delamination or spalling from the EPS concrete substrate. Discoloration around the cracks was also apparent. The crack pattern is indicative of shrinkage cracking which is generally caused by excessive evaporation of water from the latex modified concrete. In the case of the latex mortar in this test, the top surface of the overlay was exposed to ambient temperatures generally in the range of 80° to 95°F. In contrast, the bottom of the overlay was in contact with water saturated EPS concrete at a temperature probably only 55° to 65°F. Thus, the top surface dried out at a much faster rate, thereby creating the shrinkage cracks. This problem has recently been recognized by latex-modified concrete producers and the construction industry, and the most current installation procedures for latex modified cement overlays on bridge decks specify a wet cure of 48 hr in order to control shrinkage cracking.

The discoloration around the cracks appears to have been due to the percolation of groundwater up through the substrate and cracked overlay followed by evaporation from the upper surface.

The only portion of the floor that did not contain cracks was an area which was generally shielded from direct sunlight by a balcony above it. This observation tends to support the possibility that the cracking occured due to high surface moisture evaporation rates.

A survey was made of the thermal conductivity of the lightweight latex modified mortar overlay using a Shotherm QTM-D2 Quick Thermal Conductivity Meter. These data are summarized in Table 18, and they indicate an average value of

TABLE 18

Thermal Conductivity of Lightweight Latex Modified Mortar Overlay After Field Exposure

Panel No[a]	Thermal conductivity, BTU/hr-ft-°F
6	0.179
	0.190
5	0.173
	0.173
4	0.184
	0.184
3	0.177
	0.194
	0.167
2	0.194
	0.213
Mean	0.184
Std. Dev.	0.013
Coef. of Var.	7.0%

[a]refers to Figure 6.

0.184 BTU/hr-ft-°F. The coefficient of variation was 7%. Earlier measurements made on samples cast during the placement of the overlay had an average value of 0.20 BTU/hr-ft-°F.

Thermal conductivity measurements were also made on the wall containing the insulating latex modified motar overlay. In this case the average value was 0.175 BTU/hr-ft-°F, considerably lower than the control value of 0.476 BTU/hr-ft-°F for the EPS concrete covered with a cement mortar.

9. Summary

Based upon the results obtained during the installation and subsequent inspection after 9 months of field exposure of the lightweight latex modified mortar overlay, the following conclusions can be made: 1) the permeability of the insulating overlay is very low, and as a result, the overlay maintains its low thermal conductivity even in moist environmental areas, 2) the shrinkage cracking that occurred appears to be related to inadequate installation procedures rather than an inherent problem with the composite, and 3) the composite bonds well to concrete surface insulation and repair materials, thereby making it suitable for retrofit applications as well as new construction.

COMPUTER SIMULATION MODEL DEVELOPMENT

As a subcontracted effort with Robert F. Benenati, Inc. a computer software program was developed for use in the calculation of LNG boil-off rates and dispersion distances. A programmed floppy disk which can be used in a PC or equivalent type computer, and a User Manual were prepared. The program is written in the C program language. Copies of the computer code and the User Manual are given in Appendix 1 and 2, respectively.

The personal computer-based program was designed to provide the user with vaporization rate data for LNG spills within a user-defined LNG storage dike or other impoundment. By calculating solid conductive heat transfer up through up to three layers of dike floor and wall materials, the program can be used for evaluating the effectiveness of dike insulating alternatives in mitigating rapid vaporization of spilled LNG. Vaporization rates and volumes are provided to assist the user in determining hazard zones associated with downwind dispersion of the resulting LNG vapor cloud. Ideally, the user would use calculated vaporization rates as input to an appropriate heavy gas vapor dispersion model or laboratory experiment.

In addition, the program provides the user with the option of calculating vapor dispersion distances directly from the program, which includes a simple Gaussian passive dispersion procedure. However, this dispersion calculation should be used for comparative purposes only since, as typical of Gaussian dispersion models, it neglects important LNG vapor dispersion physics. Dispersion calculations produced by the program should not be used for site-specific hazard evaluation or for regulatory compliance evaluation purposes.

The objective of the User Manual is to provide program users with information on program organization and operation as well as underlying calculation approaches employed. An error in the program relating to the association of spill rates to vaporization was recently discovered. Resolution of this program was outside of the budget limitations of the contract.

CONCLUSIONS AND RECOMMENDATIONS

The results from the laboratory development and subsequent field evaluation of latex modified lightweight cement composites indicate that the materials have properties that make them suitable for use as durable load bearing insulation on containment dikes at LNG storage facilities. The composite bonds well to conventional portland cement concrete, EPS-based insulating concretes, and polymeric coatings that are sometimes placed on EPS concrete to reduce water absorption and improve its durability. As a result of the excellent bonding to these substrates, the insulating composite can be used for retrofit applications as well as new construction. A thickness of 0.75-in. will provide adequate insulation to substantially reduce LNG boil-off rates. Based upon this thickness, the cost of the materials is estimated as $\$1.00/ft^2$.

The recommended procedures for surface preparation, mixing and placement of the insulating composite are as follows:

1. The overlay must be placed on a clean and structurally sound substrate. Sandblasting or other mechanical abrading methods should be used to remove any deteriorated concrete or laitances from the substrate surface prior to application of the overlay. The degree of uniformity in the flatness of the surface will determine the amount of overlay material needed to insure a minimum thickness of 0.75-in. Therefore, any holes or irregularities in the substrate should be filled using conventional repair materials.
2. Prior to the application of the insulating overlay, the clean substrate surface should be wetted with water and then a bonding agent consisting of a 50 wt% latex - 50 wt% cement slurry applied. The slurry can be spread using brooms.
3. For large installations, continuous automated batching of the solid and liquid constituents in the insulating composite, mixing, and placement, can most economically be performed using conventional concrete industry equipment such as a concrete mobile. This will also help to insure a homogeneous

overlay. Smaller quantities can be batch mixed in conventional drum-type concrete mixers and placed by hand.

4. After placement, screeding, and surface finishing, wet curing of the composite is essential. All overlaid surfaces should be covered with a single layer of water saturated burlap immediately after the finishing operation. Then apply a single layer of polyethylene film onto the burlap before the burlap begins to dry. An alternate method is to apply a fog spray directly onto the overlay. Wet curing should be maintained for a minimum of 48 hr. Air curing until the specified strength or cure time has been achieved should then be performed.

REFERENCES

1. Chatlos, D.J. and Reid, R.C. Boiling and Spreading Rates of Instantaneous Spills of Liquid Methane on Water. GRI-81/0045, April 1982.
2. Welker, J.R. Vaporization of LNG Spills on Composite Materials, Applied Technology Corporation, OK, Sept. 1983.
3. "Evaluation of LNG Vapor Control Methods," Arthur, D. Little, Inc., Cambridge, MA, Oct. 1974.
4. Fontana, J.J., Cheng, H.C., and Reams, W. Development of an Insulating Polymer Concrete Overlay for Dike Insulation at Long Island Lighting Company's LNG Storage Facility, BNL 39906-R, June 1987.
5. Fontana, J. J. and Steinberg, M. Development of Polymer Concrete for Dike Insulation at LNG Facilities, Final Report, BNL 35689, GRI-84/0193, Nov. 1984.
6. Fontana, J. J., Cheng, H. C. Steinberg, M. Reams, W., and Elling, D. Development of Polymer Concrete for Dike Insulation at LNG Facilities, Phase II, BNL 38808-R, GRI-86/0249, Oct. 1986.
7. Fontana, J. J., Reams, W., and Elling, D. Development of Polymer Concrete for Dike Insulation at LNG Facilities, Phase III, BNL 40632, GRI-87/0301, Oct. 1987.
8. "Lightweight Concrete," American Concrete Institute Publication SP-29, Detroit, MI, 1971.

APPENDIX 1

```
/****  07/03/89     to beep a brief sound                                  */

void bleep (void)
{
sound (440);
delay (500);
nosound ();
return;
}
```

```
/****  05/24/89  to write base line on screen                    baseLine   */

#include <conio.h>
#include <stdio.h>
#define barColor textattr (BLACK + (LIGHTGRAY<<4));
#define stdColor textattr (LIGHTGRAY + (BLACK<<4));


/* function prototypes                                                      */

void baseLine (void);
void mycputs (int, int, char []);

void baseLine (void)
{
barColor;                                                          "
mycputs (1, 25, "  Wellborn Systems
        "                          copyright 1989  ");
stdColor;
return;
}  /*  baseLine                   bgen.lib   */
```

```
                                                                barText
/****  03/25/89                                                          */
      to write text on the base line

#include <conio.h>
#include <stdio.h>
#define barColor textattr (BLACK + (LIGHTGRAY<<4));
#define stdColor textattr (LIGHTGRAY + (BLACK<<4));

void barText (int x, char text[])
{
window (1, 1, 80, 25);
barColor;
mycputs (x, 25, text);
stdColor;
return;
}   /*   barText                              bgen.lib    */
```

```
/****  04/20/89  to get the absolute value of a real number  absc        */

double absv (double arg)
{
return ((arg < 0.0)? -arg:arg);
}
```

```
                                                                conduction
/****  03/18/89
      function solves the one dimension conduction equation for a fixed
      surface temperature boundary condition for up to three zones, each
      different
      ARGUMENTS:
      ts -- surface temperature, deg F
      alpha[] -- thermal diffusivity for each zone, sq ft/hr
      k[] -- thermal conductivity of each zone,BTU/hr,ft,degF
      deltime -- time step, seconds
      x[] -- vector of node lengths, ft
      t[] -- vector of node temperatures, deg F
      nc[] -- number of nodes in each zone                              */
      n -- count of total number of nodes

void conduction (double ts, double alpha[], double k[], double deltime,
                 double x[], double t[], int nc[], int n)

{
double qin, qout, term, tnew[50];
int i=0, j=0, jflag=0, getoutflag = 0, ncsum;
ncsum = nc[0];
term = 2.0 * alpha[0] * deltime;
qout = (t[0] - ts) / x[0];
in:  qin = (t[i+1] - t[i]) / (x[i] + x[i+1]);
tn:  tnew[i] = t[i] + term / x[i] * (qin - qout);
        if (getoutflag) goto getout;
        if (jflag){ term = 2.0 * alpha[j] * deltime; jflag = 0;}
        if (!qin) goto getout;
        qout = qin;
        if (++i < ncsum-1) goto in;
        if (i == n-1){ qin = 0.0;getoutflag = 1; goto tn;}
        qin = (t[i+1] - t[i]) / (x[i] + (x[i+1] * k[j]) / k[++j]);
        ncsum += nc[++j];
        jflag = 1;
        goto tn;
getout: for (j = 0; j <= i; j++){
                        t[j] = tnew[j];}

return;
}  /* conduction                                dike.lib       */
```

```
/****  03/30/89  to assign coordinate values to all nodes    coordina
          end insure that node boundaries coincide with zone
          boundaries; function returns n, the count of nodes in
          each zone.  It also returns the total count of all nodes.
          Max nodes = 100, max zones = 3.

          ARGUMENTS:
                 l1    -  length of zone one, inches
                 l2    -  length of zone 2
                 l3    -  length of zone 3
                 l[]   -  coordinate vector for all nodes,l[0]=0, l[i] = l1+l2+l
                 nc[]  -  count of nodes in each zone
                 nodePos[] - last node in zones 1 and 2

/*  function prototypes                                                 */

int coordinates (double, double, double, double[], int[], int[]);

int coordinates (double l1, double l2, double l3, double l[], int nc[]
                                    int nodePos[])
{
int i, ncsum = 0, zone = 0;
double x[] = {0.0,0.01,0.01,0.02,0.02,0.03,0.04,0.05,0.05,0.1,0.1,0.25,0.25,
                          0.5,0.5,1.0},
zoneLength;                  /* boundary of current zone, inches          */
l[0] = 0.0;
nc[2] = nc[1] = nc[0] = 0;
zoneLength = l1;
for (i = 1; i < 50; i++){
        l[i] = l[i-1] + ((i<=15) ? x[i] : 1.0);
        if (l[i] >= zoneLength){
                nc[zone] = i - ncsum;
                nodePos[zone] = i;
                ncsum += nc[zone++];
                l[i] = zoneLength;
                switch (zone){
                        case 1:
                        if (l2 == 0.0) return i;
                        zoneLength += l2;
                        break;
                        case 2:
                        if (l3 == 0.0) return i;
                        zoneLength += l3;
                        break;
                        default:
                        return i;}
                        }}
if (l[49] > zoneLength) l[49] = zoneLength;
return 50;
}   /*  coordinates                    dike.lib  */
```

```
/****  06/22/89    to turn cursor on                                    cursoron    */

#include <conio.h>
#include <dos.h>
#include <stdio.h>

/*  function prototypes                                                          */

void cursorOn (int);

void cursorOff (int start)
{
union REGS regs;
int end = 13;
regs.h.ch = (char)start;
regs.h.cl = (char)end;
regs.h.ah = 1;
int86(0x10, &regs, &regs);
return;
}  /*  cursorOn                          bgen.lib  */
```

```
exit
/****  06/22/89    to turn cursor off                                    cursroff    */

#include <conio.h>
#include <dos.h>
#include <stdio.h>

/*  function prototypes                                                             */

void cursorOff (void);

void cursorOff (void)
{
union REGS regs;
regs.h.ch = 0x20;
regs.h.ah = 1;
int86(0x10, &regs, &regs);
return;
}  /*  cursorOff                          bgen.lib  */
```

```
                                                                    main    */
/****   09/02/89

#include <conio.h>
#include <stdio.h>
#include <b:dike.h>
#include <time.h>

                                                                            */
/* function prototypes

int coordinates (double, double, double, double [], int [], int []);
void cursorOff (void);
void cursorOn (int);
void dikeDim2 (int, int);
void dikeDim3 (int, int, int, double []);
double dikeDimension (int, int, int, double, double [], double []);
void dikeMaterials (int, double *, double [][4]);
void disclaim (int, int, int far *, int);
int dispMatlPropt (int, double *, double[][4]);
void distance (char, double, double, double);
void drawAbox (int,int,int,int,char *,char *);
int far * eqpList (void);
double flashFraction (double);
void floorWall (int, double []);
int getKey (int, int, char *[], int, int);
double getNum (void);
void nodeLength (int, double [], double []);
void mycputs (int, int, char[]);
void oneLayerBoiloff (double, double, double, double, double [],
                      double *, double *);
double pipeFlow (double, double, double, double, double, double, double *);
void report (int,int,int,double,double,double[],double,double,double,
                    double[],double,double[][4],double,double,double);
void report0 (int, double, double, double);
void report1 (double);
int report2 (double);
int rerun (void);
int respond (int, int);
int spillFacts (double *, double *, double *);
void splash0 (int, int, int far *, int);
void splash1 (int far *, int);
double tank (double *, double *, double *, double *, double *, double *);
void transient (double [], double [], double [], double [], double [],
                int  [], int, int[]);
void tansient1 (double [], double [], double [], double [], double [],
                int  [], int, int[]);
void twoLayerBoilOff (double, double [], double [], double, double [],
                      double [], double *, double *);
void warning (int, double, double);
char weather (void);


main()
{
int i, x, y;
int far *videoptr;
double l1,l2,l3, floorArea;

alpha[0] = alpha[1] = alpha[2] = k[0] = k[1] = k[2] = 0.0;
videoptr = eqpList ();
splash0 (1, 1, videoptr, 0x0700);
disclaim(1, 1, videoptr, 0x0700);
splash1 (videoptr, 0x0700);
start:barText (30, "selection keys only");
```

```
time1OverDike = time2OverDike = disp1Source = disp2Source = 0.0;
shape = dikeDetails (&style, &typecon);
tankVol = tank (&tankHeight, &tankDia, &htUllage, &ullagePress, &ullageVol,

flashFrac = flashFraction (ullagePress);
dikeVol = dikeDimension (style, shape, typecon,tankDia, dimension, dikeArea);
dikeVapVol = dikeVol - tankArea * dimension[0];
if (dikeVol < tankVol) warning (1, dikeVol, tankVol);
else if (dikeVol < 1.1 * tankVol) warning (0, dikeVol, tankVol);
floorWall (typecon, thickness);
dikeMaterials (typecon, &soilMoisture, propts);

spillMode = spillFacts (&spillRate, &spillTime, &pipe_i_d);
floorArea = dikeArea[0];
if(spillMode == 2) l1 = pipeFlow (pipe_i_d, htUllage,
                          ullagePress, floorArea, tankArea, spillTime, &spillRate
if (l1 < spillTime) spillTime = l1;
windTemp (&windSpeed, tempture);
ambientTemp = tempture[0];
barText (30, "   patience     ");
mycputs (10, 12, "calculating - Please wait");
cursorOff();

report (shape, style, typecon, tankDia, tankHeight, dimension,
        htUllage, ullagePress, ambientTemp, thickness, windSpeed,
           propts, soilMoisture, tankVol, dikeVol);
report0 (spillMode, pipe_i_d, spillRate, spillTime);

ullageVol = tankArea * htUllage;     /* correct dike vapor vol and dike    */
ullageLiqHt = ullageVol / dikeArea[0];      /*  wall area due to liquid    */
dikeVapVol = dikeVol - ullageVol - tankArea *(dimension[0] - ullageLiqHt);
dikeWallArea = dikeArea[1] * ullageLiqHt / dimension[0];
dikeArea[1] = dikeWallArea;

                                                          /* calculate boiloff fro
l[0] = thickness[0];
l[1] = thickness[1];
alpha[2] = propts[0][3];
k[2] = propts[0][2];
switch (typecon){                              /* dike with liner            */
          case 1:
          case 2:
               alpha[0] = alpha[1] = propts[typecon][3];
               k[0] = k[1] = propts[typecon][2];
          doit:twoLayerBoilOff (ambientTemp, k, alpha, dikeVapVol,dikeArea, 1,
                                      &time1OverDike, &disp1Source);
               break;
          case 3:
          alpha[0] = propts[typecon][3];
               alpha[1] = propts[typecon-1][3];
               k[0] = propts[typecon][2];
               k[1] = propts[typecon-1][2];
               goto doit;
          case 4:
               alpha[0] = alpha[1] = propts[3][3];
               k[0] = k[1] = propts[3][2];
               goto doit;
          case 0:                                                 /* dike without l
        alpha[0] = propts[0][3];
        k[0] = propts[0][2];
        oneLayerBoiloff (ambientTemp, k[0], alpha[0], dikeVapVol,dikeArea,
                         &time1OverDike, &disp1Source);
        }
```

```
report1 (time1OverDike);

                                                    /* calculate boil off from insulated dik

alpha[2] = propts[0][3];
k[2] = propts[0][2];
l[0] = thickness[0] + thickness[2];
l[1] = thickness[1] + thickness[3];
switch (typecon){
        case 0:
        alpha[0] = propts[4][3];
           alpha[1] = propts[4][3];
        k[0] = propts[4][2];
           k[1] = propts[4][2];
           break;
        case 1:
        case 2:
           k[0] = l[0] / (thickness[0] / propts[typecon][2] + thickness[2] /
                                                              propts[4][2]);
           k[1] = l[1] / (thickness[1] / propts[typecon][2] + thickness[3] /
                                                              propts[4][2]);
           alpha[0] = l[0] * k[0] / (thickness[0] * propts[typecon][0] +
                                                              thickness[2] * propts[4][0])
           alpha[1] = l[1] * k[1] / (thickness[1] * propts[typecon][0] +
                                                              thickness[3] * propts[4][0])

           break;
        case 3:
           k[0] = l[0] / (thickness[0] / propts[2][2] + thickness[2] /
                                                              propts[4][2]);
           k[1] = l[1] / (thickness[1] / propts[1][2] + thickness[3] /
                                                              propts[4][2]);
           alpha[0] = l[0] * k[0] / (thickness[0] * propts[2][0] +
                                                              thickness[2] * propts[4][0])
           alpha[1] = l[1] * k[1] / (thickness[1] * propts[1][0] +
                                                              thickness[3] * propts[4][0])

           break;
        case 4:
        k[0] = l[0] / (thickness[0] / propts[3][2] + thickness[2] /
                                                              propts[4][2]);
           k[1] = l[1] / (thickness[1] / propts[2][2] + thickness[3] /
                                                              propts[4][2]);
           alpha[0] = l[0] * k[0] / (thickness[0] * propts[3][0] +
                                                              thickness[2] * propts[4][0])
           alpha[1] = l[1] * k[1] / (thickness[1] * propts[2][0] +
                                                              thickness[3] * propts[4][0])

        }
twoLayerBoilOff (ambientTemp, k, alpha, dikeVapVol, dikeArea, l,
                                                    &time2OverDike, &dis

i = report2 (time2OverDike);
if (i)goto end;
weatherMode = weather();
fprintf (stdprn, "\n\n                                    Downwind Dispersion Information"
                                                    (based on weather type %c)\n"
                        "\n
                    "\n\nFor the dike 'as built',",weatherMode);
distance (weatherMode, windSpeed, disp1Source/dimension[1], dimension[1]);
fprintf (stdprn, "\nFor the dike with insulation,");
distance (weatherMode, windSpeed, disp2Source/dimension[1], dimension[1]);
end:fprintf (stdprn, "                                                    ");
if (!rerun()) goto start;

exit(0);
}
```

```
/****   06/02/89                                                    dikedeta
       to get dike shape, style, and construction materials                  */

#include <conio.h>
#include <stdio.h>

char *shapes[2]={
                "Circular",
                "Rectangular"
                };
char *styles[3]={
                "Straight Sides",
                "Sloped Sides",
                "Sloped Sides w/shelf"
                };
char *construct[5]={
                 "Tamped Earth + Gunite",
                 "Earth + Poured Concrete",
                 "Earth,concrete floor,gunite wall",
                 "Tamped Earth",
                 "Tamped Earth + Loose Rock"
                 };

/* function prototypes                                                    */

void drawAbox (int,int,int,int,char *,char *);
int error (int);
int getKey (int, int, char *[], int, int);
void mycputs (int, int, char[]);

int dikeDetails (int *style, int *typecon)
{
int i = 5, shape, x = 19, y = 7;
window (1, 1, 52, 24);
clrscr ();
window (1, 1, 80, 25);
model();
drawAbox(x, y, 18, 2, "Dike Shapes", "Select Shape");
shape = getKey(x+2, y+3, shapes, 2, 0);
gotoxy (69, 7);
if (shape) cputs ("rectangle");
else cputs ("circle");
drawAbox(x, y, 22, 3, "Dike Styles", "Select Style");
*style = getKey(x+2, y+3, styles, 3, 0);
gotoxy (69,8);
switch (*style){
        case 0:
        cputs ("straight");
        i = 3;
        break;
        case 1:
        cputs ("sloped");
        break;
        case 2:
        cputs ("sl/shelf");
        break;}
drawAbox(x-2, y, 34, i, "Dike Construction", "Select Construction Type");
*typecon = getKey(x, y+3, construct, i, 0);
gotoxy (69,9);
switch (*typecon){
        case 3:
        cputs ("earth");
        break;
```

```
        case 4:
        cputs ("e+rock");
        break;
        case 0:
        cputs ("e+gunite");
        break;
        case 1:
        cputs ("e+concrete");
        break;
        case 2:
        cputs ("conc+gunite");
        break;}
window (1, 1, 52, 24);
clrscr ();
return shape;
}    /*   dikeDetails                          dike.lib  */
```

```
/****    . 03/09/89   to paint part of dike dimension screen   dikeDim2   */
#include <conio.h>
#include <dos.h>
#include <math.h>
#include <stdio.h>

/* function prototypes                                                    */

void dikeDim2 (int, int);

void dikeDim2 (int y, int style)
{
gotoxy (5, y);
cputs ("Angle(degrees from vertical) = ?");
if (style != 2) return;
gotoxy(5, y+2);
cputs ("Shelf height, inches = ?");
gotoxy (5, y+4);
cputs ("Shelf width, inches = ?");
return;
}  /*  dikeDim2   */
```

```
/***  09/13/89  to get wall angle and shelf dimensions       dikedim3

  function prototypes                                                    */

void cleanSpace (int, int, int);
void dikeDim3 (int, int, int, double[]);
double getNum (void);

#include <stdio.h>

void dikeDim3 (int y, int style, int typecon, double dimension[])
/*       dimension[3] = angle(radians) from vertical, read in as degrees
                        [4] = shelf height, feet
                        [5] = shelf width
{
double angle;
redo: gotoxy (36, y);
angle = getNum ();
if ((typecon == 3 | typecon == 4) && angle < 52.0 && error (4)) goto gogo;
if (angle >= 80.0 && error (2)){
        gogo: cleanSpace (34, y, 9);
        goto redo;}
gotoxy (70, 13);
cprintf ("%.1f", angle);
dimension[3] = angle / 57.296;
if (style != 2){
        mycputs (69, 14, " - - - ");
        return;}
gotoxy (28, y+2);
dimension[4] = getNum ();
gotoxy (27, y+4);
dimension[5] = getNum ();
gotoxy (69, 14);
cprintf ("%.1fx%.1f", dimension[4], dimension[5]);
dimension[4] /= 12.0;
dimension[5] /= 12.0;
return;
}  /* dikeDim3                    dike.lib  */
```

```
/***    09/25/89                                                    dikedimn
       to get dike dimensions & compute & return the dike volume          */

#include <conio.h>
#include <dos.h>
#include <math.h>
#include <stdio.h>
#define PI 3.14159

/* function prototypes                                                    */

double dikeDimension (int, int, int, double, double [], double []);
int error (int);
double getNum (void);

double dikeDimension (int style, int shape, int typecon, double tankDia,
                      double dimension[], double dikeArea[] )
/*   dimension[0] = dike height, feet
               [1] = diameter/length
               [2] = width
               [3] = angle of wall from vertical, radians
               [4] = shelf height
               [5] = shelf width
        dikeArea[0] = floor area(not including tank), sqft
             [1] = wall area                                           */
{
double area1,        /* total area of floor of dike, sq ft              */
         area2,      /* total area of top of dike, sq ft                   */
         bigDia,     /* top dia of tapered circular dike, feet             */
       shelfVol,     /* volume of shelf at floor, cu ft                 */
         smallDia,   /* diameter inside shelf at floor for style=2          */
         vol,        /* dike volume, cu ft                                 */
         xtra1;   /* avg increase in length & width due to tapered walls   */

gotoxy (13, 1);
if (shape) cputs ("Rectangular ");
else cputs ("Circular ");
cputs ("Dike Dimensions");
please (10, 22);
mycputs (5, 5, "Height, ft = ?");
gotoxy (5, 7);
switch (shape){
        case 0:                                          /* circular      */
                cputs ("Diameter (at floor), ft = ?");
                if (style != 0) dikeDim2 (9, style);
                break;
        case 1:                                          /* rectangular   */
                cputs ("Length, ft = ?");
                mycputs (5, 9, "Width, ft = ?");
                if (style != 0) dikeDim2 (11, style);
                break;}
gotoxy (18, 5);
dimension[0] = getNum ();           /* get numeric values of dimensions */
gotoxy (70,10);
cprintf ("%.1f", dimension[0]);
switch (shape){
        case 0:                                  /*  circular             */
                mycputs (56, 11, "diameter");
                redo: gotoxy (31, 7);
                        dimension[1] = getNum ();
                        if ((tankDia >= dimension[1]) && (error (0))){
                              cleanSpace (20, 7, 9);
                              goto redo;}
```

```
                    gotoxy (70, 11);
                    cprintf ("%.1f", dimension[1]);
                    vol = 0.786 * dimension[0] * dimension[1] * dimension[1
                    if (style != 0){dikeDim3 (9, style, typecon, dimension)
                          bigDia = 2.0 * dimension[0] * tan (dimension[3])

                          vol = (PI / 12.0) * dimension[0] * (dimension[1]
                                dimension[1] + bigDia * bigDia + sqrt (

                    if (style == 2){
                          smallDia = dimension[1] - dimension[5] / 6.0;
                          /* 0.06545 = pi / (4 * 12) */
                          shelfVol = 0.06545 * (dimension[1] * dimension[1
                                                smallDia * smallDia) *

                          vol -= shelfVol;}
               break;
      case 1:
               redo1: gotoxy (18, 7);
               dimension[1] = getNum ();
               gotoxy (17, 9);
               dimension[2] = getNum ();
               if ((tankDia >= dimension[1]) || (tankDia >= dimension[2]) &&

                       cleanSpace (18, 7, 9);
                       cleanSpace (17, 9, 9);
                       goto redo1;}
               if (dimension[2] > dimension[1]){  /* set largest dim = length *
                       xtra1 = dimension[1];
                       dimension[1] = dimension[2];
                       dimension[1] = xtra1;}
               mycputs (56, 11, "length ");
               gotoxy (70, 11);
               cprintf ("%.1f", dimension[1]);
               gotoxy (70, 12);
               cprintf ("%.1f", dimension[2]);
               vol = dimension[0] * dimension[1] * dimension[2];
               if (style != 0){ dikeDim3 (11, style, typecon, dimension);
                       xtra1 = 2.0 * dimension[0] * tan (dimension[3]);
                       area1 = dimension[1] * dimension[2];
                       area2 = (dimension[1] + xtra1) * (dimension[2] + xtra1);
                       vol = 0.3333 * (area1 + area2 + sqrt (area1 * area2)) *

               if (style == 2){                                    /* 0.013899 = 2
                       shelfVol = 0.013889 * dimension[5] * dimension[4] *
                                                          (dimension[1] + dimens

                       vol -= shelfVol;}
               break;}
                                /*  calculate floor and wall areas   */
switch (shape){
       case 0:
               dikeArea[0] = 0.786 * (dimension[1] * dimension[1] - tankDia *

               dikeArea[1] = PI * dimension[1] * dimension[0];
               break;
       case 1:
               dikeArea[0] = dimension[1] * dimension[2] - 0.786 * tankDia *

               dikeArea[1] = dimension[0] * 2.0 * (dimension[1] + dimension[2])
               break;}
window (1, 1, 52, 24);
clrscr ();
window (1, 1, 80, 25);
return vol;
```

```
}   /*  dikeDimension                    dike.lib     */
```

```
                                                                dikemate
/****   09/14/89
      to get soil moisture content and dimensions and properties of dike
         construction materials
                                                                        */

   function prototypes

void barText (int, char *);
void dikeMaterials (int, double *, double[][4]);
int dispMatlPropt (int, double *, double[][4]);
void getPropts (int, double *, double[][4]);

#include <conio.h>

void dikeMaterials (int typecon, double *soilMoisture, double propts[][4])
{
double x;              /* dummy place holder for soil moisture in functions */
barText (30, "selection keys only");
window (1, 1, 52, 24);
mycputs (12, 1, "Dike Material Properties");
if (dispMatlProp (2, soilMoisture, propts)) getPropts (2, soilMoisture,
                                                          propts);
switch (typecon){       /* 3 earth, 0 gunite, 1 concrete, 2 c+g, 4 rock  */
        case 3:
            break;
        case 0:
                if (dispMatlProp (0, &x, propts)) getPropts (0, &x, propts);
                break;
        case 1:
                if (dispMatlProp (1, &x, propts)) getPropts (1, &x, propts);
                break;
        case 2:
                if (dispMatlProp (0, &x, propts)) getPropts (0, &x, propts);
                if (dispMatlProp (1, &x, propts)) getPropts (1, &x, propts);
                break;
        case 4:
                if (dispMatlProp (3, &x, propts)) getPropts (3, &x, propts);
                break;}             /*now get insulation properties          *
 if (dispMatlProp (4, &x, propts)) getPropts (4, &x, propts);
 clrscr ();
 return;
 }  /*  dikeMaterials                    dike.lib  */
```

```
/****  09/6/89  to write the GRI disclaimer to the video    disclaim
          ram directly                                                */

#include <stdio.h>
#include <conio.h>

/*  function prototypes                                              */

void disclaim (int, int, int far *, int);
void pakc (int, int, int far *, int);
void splash (int, int, int far *, char [], int);

void disclaim (int x, int y, int far *videoptr, int colr)
{
char msg [] = "                          "
        "G R I  DISCLAIMER\n\n\nLEGAL NOTICE  This program was "
        "prepared by Wellborn Systems\nas a project sponsored by the Gas"
        " Research Institute (GRI).\nNeither GRI, members of GRI, nor any"
        " person acting on behalf\nof either:\n\na.  Makes any warranty or"
        " representation, express or\n    implied, with respect to the "
        "accuracy, completeness,\n    or usefullness of the information "
        "contained in this\n    computer program, or that the use of any "
        "apparatus,\n    method, or process disclosed in this program may not"
        "\n    infringe privately owned rights; or\n\nb.  Assumes any "
        "liability with respect to the use of, or\n    for damages resulting"
        " from the use of, any\n    information, apparatus, method, or process"
        " disclosed\n    in this program.";

splash (x, y, videoptr, msg, colr);
splash (1, 25, videoptr, " Gas Research Institute", 0x7000);
pakc (24, 24, videoptr, colr);
window (1, 1, 80, 24);
clrscr ();
return;
}  /*  disclain                          dike.lib  */
```

```
/*** 09/13/89  to display material properties and request    dispmpro
        approval for their use.  Returns 0 if properties are
        acceptable; 1 if user wishes to substitute other values.
                                                                    */
  function prototypes

void cleanSpace (int, int, int);
int dispMatlProp (int, double *, double[][4]);
void mycputs (int, int, char[]);
int respond (int, int, char[]);

#include <stdio.h>

int dispMatlProp (int matCode, double *soilMoisture, double propts[][4])
{

char *materials[5] ={" gunite      ",
                     " concrete    ",
                     " tamped earth",
                     " loose rock  ",
                     " insulation  "};

int i = 0, j;
gotoxy (17, 3);
cputs (materials[matCode]);
mycputs (5, 5, "The approximate properties of ");
cputs (materials[matCode]);
mycputs (5, 6, "are as follows");
mycputs (5, 7, "density-------------          #/cu ft");
gotoxy (26, 7);
printf ("%.3f", propts[matCode][0]);
mycputs (5, 9, "heat capacity--------          BTU/#-degF");
gotoxy (26,9);
printf ("%.3f", propts[matCode][1]);
mycputs (5, 11, "thermal conductivity-          BTU/hr-ft-degF");
gotoxy (26, 11);
printf ("%.3f", propts[matCode][2]);
gotoxy (5,13);
switch (matCode){
        case 2:
                printf ("moisture content-----%.1f       #/#dry soil",
                                *soilMoisture);
                mycputs (5, 15, "however these values can vary locally");
                mycputs (5, 16, "due to moisture and other factors");
                break;
        case 3:
                cputs ("loose rock characteristics depend on the    ");
                mycputs (5, 14, "size distribution of the rock mixture");
                break;
        case 0:
                cputs ("gunite is known to vary from one application");
                mycputs (5, 14, "to another.");
                break;
        case 1:
                cputs ("concrete properties vary somewhat with       ");
                mycputs (5, 14, "pouring practice");
                break;
        case 4:
                cputs ("insulation is a highly variable product     ");
                break;}
 if (respond (10, 17, "Are these values acceptable?")){
        cleanSpace (25, 7, 9);
        cleanSpace (25, 9, 9);
        cleanSpace (25,11, 9);
```

```
        if (!matCode) cleanSpace (25, 13, 9);
        i = 1;}
window (5, 14, 52, 20);
clrscr ();
window (1, 1, 52, 24);
return i;
}  /*  dispMatlProp                          dike.lib    */
```

```
/****  06/01/89  to compute the maximum downwind distance    distance
          at which a methane concentration of 2.5% will be found
          ARGUMENTS:
          weatherMode -- letter B thru F for Gifford atmospheric categories
          windVel        ft/sec
          source  -- flow over dike, #/sec/ft of dike width                */

#include <math.h>
#include <stdio.h>
                                                                           */
/*  function prototypes

double absv (double);
void distance (char, double, double, double);
double erf (double);

void distance (char weather, double windVel, double source, double dikeW)
{
double con, con1, con2, con3,              /* Giffords categories          */
dist1 = 0.0,                               /* dist with conc > 2.5%        */
dist2,                                     /* dist with conc < 2.5%        */
xlee = 100.0,                              /* new trial dist, ft           */
newConc,                                 /* calculated methane conc at xlee */
power = 0.919,
                                 /* dispersion factors                     */
sigy,sigz,                       /* temporary store                        */
term,
ystar, zstar;

int split = 0;

if (source == 0.0){
        fprintf (stdprn, "there is no downwind dispersion for this case\n"
                                  "in the first 45 minutes.");

        return;}

switch (weather){
        case 'B':
                con = 158.0;
                con1 = 2.041;
                con2 = 1.048;
                con3 = 0.041;
                power = 0.9;
                break;
        case 'C':
                con = 104.0;
                con1 = 1.786;
                con2 = 0.914;
                con3 = 0.0;
                power = 0.913;
                break;
        case 'D':
                con = 69.0;
                con1 = 1.505;
                con2 = 0.737;
                con3 = -0.105;
                break;
        case 'E':
                con = 51.0;
                con1 = 1.332;
                con2 = 0.678;
                con3 = -0.112;
                break;
        case 'F':
```

```
                con = 34.0;
                con1 = 1.146;
                con2 = 0.65;
                con3 = -0.113;
                break;}
redo: term = xlee / 3280.0;
sigy = 3.2808 * con * pow (term, power);
term = log10 (term);
sigz = 3.2808 * pow (10.0, con1 + term * (con2 + term * con3));
ystar = erf (dikeW / (2.8284 * sigy));    /* 2.8284 = 2*sqrt(2)          */
zstar = 0.79788456 / sigz;                /* .79788456 = 2/sqrt(2pi)     */
                                                          /* 0.6233 = 35
newConc = 0.6233 * source * ystar * zstar / (windVel * dikeW);
switch (split){
        case 0:
                if (newConc > 2.5){
                        dist1 = xlee;
                        xlee *= 2.0;
                        goto redo;}
                else{
                        dist2 = xlee;
                        split = 1;
                        new: xlee = 0.5 * (dist1 + dist2);
                        goto redo;}
        case 1:
                if (absv (newConc / 2.5 - 1.0) <= 0.0001){
                   fprintf (stdprn, " the maximum downwind distance at which the
                            "methane concentration reaches 2.5%% is %.0f ft\n",xle
                   return;}
                if (newConc > 2.5){
                        dist1 = xlee;
                        goto new;}
                else{
                        dist2 = xlee;
                        goto new;}
                }
}   /*  distance                    dike.lib  */
```

```
/****  09/07/89  to draw a single line border box of          drawbord
                specified size at x,y location
                                                                    */
  function prototypes

void drawBorder (int, int, int, int);
void mycputs (int, int, char[]);

void drawBorder (int x, int y, int width, int lines)
{
char line[80],
       line2[80];
int i;

for (i = 1, line[0] = '', line2[0] = ''; i < width; i++){
        line[i] = '';
        line2[i] = ' ';}
line[width] = '';
line2[width] = '';
line[width + 1] = line2[width + 1] = '\0';
mycputs (x, y, line);
for (i = 1; i < lines - 1; i++){
        mycputs (x, y + i, line2);}
line[0] = '';
line[width] = '';
mycputs (x, y + lines - 1, line);
return;
}   /*  drawBorder                              dike.lib  */
```

```
/****  08/10/89  to draw a box with arbitrary borders        draw box
                              at an arbitrary location
          ARGUMENTS:
          style  constant zero thru three
                        0 = single line,    1 = double line,
                        2 = single horizontal, double vertical
                        3 = double horizontal, single vertical
          x,y    screen coordinates upper left corner
          width  box width
          height box height
          color  color attribute

function prototypes                                                    */

void drawLine (int,int,int,int,int);
void drawBox (int,int,int,int,int,int);

void drawBox (int style, int x, int y, int width, int height, int color)
{
static int styles[4][4] = {{11,81,81,31},
                                                    {14,83,83,34},
                                                    {12,83,83,32},
                                                    {13,81,81,33}};

if (width * height == 0) return;
drawLine (styles[style][0], x, y++, width, color);
drawLine (styles[style][1], x, y, height - 1, color);
drawLine (styles[style][2], x + width-1, y, height - 1, color);
drawLine (styles[style][3], x, y + height - 2, width, color);
return;
}   /* drawBox                  john.lib  */
```

```
/**** 10/08/89    to clear an area of screen                    drawcler
        ARGUMENTS:
        x,y       screen coordinates ul corner
        width     width of rectangular area
        height    height of rectangular area
        color     color to be used
        dec       fill character (if any)
                                                                      */
  function prototypes

void drawClear (int, int, int, int, int, int);

#include <global.h>

void drawClear (int x, int y, int width, int height, int color, int dec)
{
register int i, j;
int far *vp;
vp = OA(x,y);
for (j = 0; j < height; j++){
        for (i = 0; i < width; i++) *(vp+i) = dec | color;
        vp += 80;}                          /*   end for  */

return;
}       /*  drawClear                     bgen.lib   */
```

```
#include <stdio.h>
/**** 07/21/89   to draw a line at arbitrary location        drawline
        ARGUMENTS:
        videoptr  indicates monochrome or color monitor
        i         = 0/1 for single/double horizontal line
                        = 1/2 for single/double vertical lineline
        startx    screen x location for start of line
        starty    screen y location for start of line
        length    total line length

  function prototypes                                                  */

void drawLine (int far *, int, int, int, int);

void drawLine(int far *videoptr, int i, int startx, int starty, int length)
{
int far *videonow;
int code[] ={196, 205, 179, 186}, x;
videonow = videoptr;
videonow += (80 * (starty - 1) + startx - 1);
switch (i){
        case 0:                             /* horizontal line          */
        case 1:
                for (x = 1; x <= length; x++){
                        *(videonow++) = code[i] | 0x0700;}
                break;
        case 2:                              /*  vertical line           */
        case 3:
                for (x = 1; x <= length; x++){
                        *(videonow) = code[i] | 0x0700;
                        videonow += 80;}
                break;}
return;
}  /*  drawLine                        bgen.lib  */
```

```
/**** 05/06/89  to check eqplist word and reset video mode   eqplist     */

#define EQLIST 0x410

/*  function prototypes                                                   */

int far * eqpList (void);

int far * eqpList (void)
{
int far *farptr;
int far *videoptr;
unsigned int eq;

farptr = (int far *)EQLIST;
eq = *farptr;
if ((eq >> 14) < 1){
        puts ("This program requires a printer attached to parallel\nprinter"
                " port LPT1");
        puts("ABORTING");
        exit (0);}

/*  should add a check for math coprocessor here                          */

switch ((eq >> 4) & 3){
        case 1:
                *farptr += 9;
        case 2:                         /* color graphics adapter        */
                videoptr = (int far *)0xB8000000;
                goto out;
        case 3:                         /* monochrome adapter            */
                videoptr = (int far *)0xB0000000;
                out:return videoptr;}
} /*  eqpList                 bgen.lib   */
```

```
/****  03/12/89                                                    erf
      returns the value of the error function                          */

#include <math.h>
#include <stdlib.h>

double erf (double);

double erf (double x)
{
#define E1 0.254829592
#define E2 -0.284496736
#define E3 1.421413741
#define E4 -1.453152027
#define E5 1.061405429
#define P 0.3275911

double t;
if (x >= 3.6) return 1.0;
if (x <= -3.6) return -1.0;
t = 1.0 / (1.0 + P * x);
return (1.0 - t * (E1 + t * (E2 + t * (E3 + t * (E4 + t * E5)))) /
        exp (x * x));
}  /* erf  */

double erf(double);
double erfc(double);

double erfc (double x)
{
return (1.0 - erf(x));
}  /*  erfc  */
```

```
/****  09/13/89  to write error msg to screen and                    error
                              solicit a response

#include <stdlib.h>
                                                                        */
/*  function prototypes

void bleep (void);
int error (int);
void mycputs (int, int, char *);
int question (int, int, char[], char[]);

int error (int no)
{
char buffer[22*11*2], line1[] = "",
                      line2[] = "                    ";

int action, i;

bleep ();
gettext (17, 14, 38, 24, buffer);
mycputs (17, 14, line1);
for (i = 1; i < 10; i++){
        mycputs (17, 14 + i, line2);}
line1[0] = '';
line1[21] = '';
mycputs (17, 24, line1);
switch (no){
        case 0:
                mycputs (18, 15, "The tank diameter");
                mycputs (18, 16, "exceeds the dike");
                mycputs (18, 17, "dimensions");
                act: action = question (19, 19, " abort ", " redo ");
                        break;
        case 1:
                mycputs (18, 15, "The liquid height");
                mycputs (18, 16, "cannot be greater");
                mycputs (18, 17, "than the tank height");
                goto act;
        case 2:
                mycputs (18, 15, "You specified a wall");
                mycputs (18, 16, "angle which is too");
                mycputs (18, 17, "shallow");
                goto act;
        case 3:
                mycputs (18, 15, "You pressed RETURN");
                mycputs (18, 16, "without entering");
                mycputs (18, 17, "any number");
                goto act;
        case 4:
                mycputs (18, 15, "Your wall angle is");
                mycputs (18, 16, "steeper than angle");
                mycputs (18, 17, "of repose");
                goto act;
        default:
                break;
                }
  if (action == 0) abort ();
  puttext (17, 14, 38, 24, buffer);
  return 1;
  }   /* error                          dike.lib  */
```

```
/****  03/07/89  to calculate the flash fraction based on     flashfra
          the ullage pressure using table look up of table B2
          (p74) of ADL report 80406, April 1978
                                                                      */
  function prototypes

double flashFraction (double);
double fun1ext (double, double [], double [], int);

double flashFraction (double ullagePress)
{
double pvt[] = {1.0,1.203,1.497,2.248,3.22,4.488,6.074,7.8},
          f[] = {0.0,0.015,0.034,0.071,0.111,0.151,0.191,0.234};

return (fun1exr (ullagePress, ptv, f, 7);
}  /*  flashFraction                        dike.lib  */
```

```
/****  03/07/89  to calculate the flash fraction based on    flashfra
          the ullage pressure using table look up of table B2
          (p74) of ADL report 80406, April 1978

  function prototypes                                                  */

double flashFraction (double);
double fun1ext (double, double [], double [], int);

double flashFraction (double ullagePress)
{
double pvt[] = {1.0,1.203,1.497,2.248,3.22,4.488,6.074,7.8},
          f[] = {0.0,0.015,0.034,0.071,0.111,0.151,0.191,0.234};

return (fun1exr (ullagePress, ptv, f, 7);
}  /*  flashFraction                    dike.lib  */
```

```
/****  09/13/89   to get dike floor covering, wall cover     floorwal
                        and insulation thickness
          ARGUMENTS: typecon  3-tamped earth
                              0-gunite floor and walls
                              1-concrete floor and walls
                              2-concrete floor,gunite walls
                              4-loose rock floor and walls
                     thickness[0] = floor covering, inches
                              [1] = wall covering
                              [2] = insulation on floor
                              [3] = insulation on walls
                                                                    */
  function prototypes

void barText (int, char []);
void floorWall (int, double []);
double getNum (void);
void mycputs (int, int, char []);

#include <stdio.h>

void floorWall (int typecon, double thickness[])
{
char *material[4] = {"  gunite    ",
                     " concrete  ",
                     "insulation ",
                     "loose rock "};

int y = 5;
barText (30, "numerics only  ");
mycputs (15, 1, "dike Liner Information");
switch (typecon){
   case 3:
        insul: mycputs (5, y, "new insulation thickness on floor,inches = ?");
        mycputs (30, y + 2, "on walls,inches = ?");
        break;
                                          /*  loose rock          */
   case 4:
        put: mycputs (5, y, "thickness of ");
        mycputs (18, 5, material[typecon - 1]);
        mycputs (29, 5, " on floor,inches = ?");
        mycputs (29, 7, " on walls,inches = ?");
        gogo: y = 9;
        goto insul;
                                          /*  gunite              */
   case 0:                                /*  concrete            */
   case 1:
        goto put;
   case 2:                  /* concrete on floor with gunite on walls     */
        mycputs (5, y, "thickness of concrete on floor,inches = ?");
        mycputs (18, 7, material[typecon-1]);
        mycputs (29, 7, "on walls,inches = ?");
        goto gogo;}

  switch (typecon){
     case 3:
          get: gotoxy (48, y);
          thickness[2] = getNum ();
          gotoxy (71, 17);
          cprintf ("%.1f", thickness[2]);
          gotoxy (48, y + 2);
          thickness[3] = getNum ();
          gotoxy (71, 18);
          cprintf ("%.1f", thickness[3]);
          break;
      case 0:
```

```
  case 1:
  case 2:
  case 4:
        gotoxy (48, 5);
        thickness[0] = getNum ();
        gotoxy (71, 15);
        cprintf ("%.1f", thickness[0]);
        gotoxy (48, 7);
        thickness[1] = getNum ();
        gotoxy (71, 16);
        cprintf ("%.1f", thickness[1]);
        goto get;}

window (1, 1, 52, 24);
clrscr ();
window (1, 1, 80, 24);
return;
}       /*  floorWall                              dike.lib  */
```

```
/***  08/31/89  table look-up with linear slope                    funlext
      interpolation and extrapolation.
      ARGUMENTS:
      x       look-up argument, double
      x1[]    table to be looked at, float
      y1[]    response table, float
      n       size of tables, int
                                                                    */

  function prototypes

double funlext (double, float [], float [], int);

double funlext (double x, float x1[], float y1[], int n)
{
int i;
if (x < x1[0]) return (y1[0]-(x1[0]-x)*(y1[1]-y1[0])/(x1[1]-x1[0]));
if (x > x1[n]) return (y1[n]+(x-x1[n])*(y1[n]-y1[n-1])/(x1[n]-x1[n-1]));
for (i=0; i<=n; i++){
        if (x == x1[i]) return y1[i];
        if (x < x1[i]) return (y1[i] - (x1[i] - x) * (y1[i] - y1[i-1]) /
                                                     (x1[i] - x1[i-1]));}

}   /* funlext              bmath.lib  */
```

```
                                                                    getKey
/*      05/26/89                                                          */
        to control menu display and return user response
#include <conio.h>
#include <dos.h>
#include <stdio.h>
#define reverseColor textattr (BLACK + (LIGHTGRAY<<4));
#define stdColor textattr (LIGHTGRAY + (BLACK<<4));
                                                                          */
/* function prototypes

int getKey (int, int, char *[], int, int);
void bleep (void);

int getKey(int x, int y, char *words[], int number, int txt)
{                                       /* line = index to menu item   */
int key, key2, line, ystart;
ystart = y;
for (line = 0; line < number; line++){
        gotoxy(x,y++);
        cprintf(words[line]);}
gotoxy(x, y = ystart);
line = 0;
reverseColor;
cprintf(words[line]);
stdColor;
while((key = getch()) != '\r'){
        if (key == 0){
                key2 = getch();
                switch (key2){
                        case 16:                /* abort */
                                exit (0);
                        case 72:                /* go up */
                                if (line == 0){
                                        bleep ();
                                        break;}
                                gotoxy(x,y--);
                                cprintf(words[line--]);
                                test: if (txt) text (line);
                                break;
                         case 80:               /* go down  */
                                if (line == number-1){
                                        bleep ();
                                        break;}
                                gotoxy(x,y++);
                                cprintf(words[line++]);
                                goto test;
                         default:
                                bleep ();
                                break;
                                }
                        }
                reverseColor;
                gotoxy(x,y);
                cprintf(words[line]);
                stdColor;
                }
  return line;
  }  /* getKey                       bgen.lib   */
```

```
/****  10/08/89  to get line lenght for a piect of text      getlen
          ARGUMENTS:
          ptx         pointer to text
          w           width of window space
          RETURN      index of next character to print

  function prototypes                                                    */

int getLen (char *, int);

int getLen (char *ptx, int w)
{
int newlen = 0;
while ((newlen < w && ptx[newlen] != '\r') && (newlen < w &&
                                                                    ptx[ne
        newlen++;
if ((ptx[newlen] != '') && (ptx[newlen] != '\r') && (ptx[newlen] != ' '))
        while (ptx[--newlen] != ' ');

        return (newlen + 1);
}
```

```
                                                                    getNum
/****      05/26/89                                                                  */
     get a number from the keyboard in string format, check for validity,
    then return a numeric value

#include <math.h>
#include <conio.h>
#include <stdio.h>
#include <dos.h>
                                                                                    */

/*  function prototypes

void bleep (void);
int error (int);
double getNum (void);
int warning (int);

double getNum (void)
{
int digitCount = 0,x,y,
         Nflag = 0,          /* change to 1 when first digit has been posted    */
         Pflag = 0;      /* change to digitCount when period has been posted    */
char numString[10] = " ",
          key;
double numb;
x = wherex();
y = wherey();
redo:while ((key = getch()) != '\r'){
        if (key == 0){ key = getch ();
                if (key == 16){ exit (0);}
                else{
                bleep();
                goto redo;}}
        switch (key){
        case  '\b' :
                if (!Nflag){ bleep(); break;}
                printf("\x1B[D");
                putch(' ');
                printf("\x1B[D");
                numString[digitCount-1] = '\0';
                if (Pflag == digitCount--)Pflag = 0;
                if (digitCount == 0)Nflag = 0;
                break;
         case '.':
                if (Pflag){ bleep(); break;}
                Pflag = digitCount;
                if (Nflag)goto postIt;
                        putch ('0');
                        numString[digitCount++] = '0';
                        Nflag = 1;
                        goto postIt;
         case '0':
                if (!Nflag){ bleep(); break;}
                postIt: putch (key);
                numString[digitCount++] = key;
                break;
         case '1':
         case '2':
         case '3':
         case '4':
         case '5':
         case '6':
         case '7':
         case '8':
```

```
        case '9':
                if (!Nflag) Nflag = 1;
                goto postIt;
        default: bleep(); break;}
                                }
if (digitCount == 0 && error (3)) {gotoxy (x, y); goto redo;}
numb = atof (numString);
if (numb == 0.0 && warning (2)) {gotoxy (x, y); goto redo;}
return (numb);
}  /*  getNum                          bgen.lib    */
```

```
/****      09/14/89                                              getpropt
          to get density, heat capacity, and thermal conductivity of
          specific dike materials and soil moisture content                */

/*  function prototypes                                                      */

void barText (int, char *);
double getNum (void);
void getPropts (int, double *, double[][4]);

void getPropts (int matl, double *soilMoisture, double propts[][4])
{
barText (30, "numerics only      ");
gotoxy (26, 7);
propts [matl][0] = getNum ();
gotoxy (26, 9);
propts[matl][1] = getNum ();
gotoxy (26, 11);
propts[matl][2] = getNum ();
propts[matl][3] = propts[matl][2] / (propts[matl][0] * propts[matl][1]);
if (matl == 2){
        gotoxy (26, 13);
        *soilMoisture = getNum ();}
barText (30, "selection keys only");
return;
}   /*   getPropts   */
```

```
/*** 10/04/89  splash a limited number of lines on screen    lmtsplsh
        note: there is no check for limits of text to be
        splashed relative to size of window
        ARGUMENTS:
        lines     count of number of lines to be splashed
        x, y      screen coordinates of up-left corner
        ch        pointer to first character of string to be splashed

  function prototypes                                                  */

char * limitSplash (int, int, int, int far *, char *);

char * limitSplash (int lines, int x, int y, int far *videoptr, char *ch)
{
int count = 0, attr = 0x07;
int far *videonow;
int far *videostart;
char c;

videonow = videostart = videoptr + x - 1 + 80 * (y - 1);
while (*ch != ''){
        c = *ch++;
        switch (c){
                case '\n':
                        if (++count == lines) return ch;
                        videonow = (videostart += 80);
                        break;
                case '{':                       /* start intensified        *
                        attr = attr^0x08;
                        break;
                case '}':                       /* return to normal         *
                        attr = 0x07;
                        break;
                case '[':                       /* start reverse video      *
                        attr = (attr&0x88)|0x70;
                        break;
                case ']':                       /* start blinking           *
                        attr = attr^0x80;
                        break;
                case '_':                       /* start underline          *
                        attr = (attr&0x88)|0x01;
                        break;
                default:
                        *(videonow++) = c|attr<<8;
                                                        }    /* end switch  */
                                                        }    /* end while   */
return 0;
}  /* limitSplash              bgen.lib                 */
```

```
                                                                    model
/****  09/25/89
      to display entire model structure on splash screen                    */

#include <conio.h>
#include <stdio.h>
                                                                            */
/*  function prototypes

void drawBorder (int, int, int, int);
void model (void);

void model (void)
{
drawBorder (53, 1, 27, 24);
mycputs (58, 1, "PROBLEM DESCRIPTION");
mycputs (55, 2, "TANK:");
mycputs (56, 3, "diameter       ft");
mycputs (56, 4, "height       ft");
mycputs (55, 5, "Liquid height       ft");
mycputs (60, 6, "pressure       psig");
mycputs (55, 7, "DIKE:shape");
mycputs (60, 8, "style");
mycputs (56, 9, "construction");
mycputs (56, 10, "height       ft");
mycputs (56, 11, "dia/length       ft");
mycputs (56, 12, "width       ft");
mycputs (56, 13, "wall angle       deg");
mycputs (56, 14, "shelf       inch");
mycputs (56, 15, "floor liner       inch");
mycputs (56, 16, "wall liner       inch");
mycputs (56, 17, "IPC on floor       inch");
mycputs (56, 18, "IPC on wall       inch");
mycputs (55, 19, "SPILL rate       gpm ");
mycputs (61, 20, "time       min");
mycputs (55, 21, "WEATHER       degF");
mycputs (55, 22, "Wind speed       m/hr");
mycputs (55, 23, "Downwind dist       ft");
return;
}   /*  model                                dike.lib  */
```

```
/****  03/27/89                                                    mycputs
            to write an arbitrary message at an arbitrary location      */

void mycputs (int x, int y, char msg[])
{
gotoxy (x, y);
cputs (msg);
}
```

```
/****  06/13/89  to print C listings with line numbers       niceprnt

Prints standard input to standard output after adding line numbers,
truncating to 80 characters, etc                                    */

#include <stdio.h>
#define min (x, y) (x < y) ? x : y

/*  function prototypes                                             */

int main ();
void nesting (unsigned *, char *);

/*  MAIN - read from STDIN one line at a time.  Reprint each line to
                    STDOUT after adding line numbers and nesting levels

main ()
{
char string[256];
unsigned linenum, level, newlevel;

linenum = 0;
newlevel = 0;
while (gets (string)){
        level = newlevel;
        nesting (&newlevel, string);
        string[70] = '\0';
        printf ("%3u[%2u]: ", ++linenum, (level<newlevel) ? level:newlevel);
        puts (string);
};

while (linenum++ % 66)
        printf ("\f\n");
}
/*** nesting - search the given string for "{" and "}". Increment nesting
                            level on "{" and decrement it on "}".

void nesting (unsigned *levelptr, char *stringptr)
{
do{
        if (*stringptr == '{')
                *levelptr +=1;
        if (*stringptr == '}')
                *levelptr -=1;
        } while (*stringptr++);
}
```

```
/****  04/27/89  boiloff from single homogeneous material    onelboff
      according to the formula t=ts+(t0-ts)*erf(X) where X=x/2t
      ARGUMENTS
      to       ambient temperature, deg F
      k        thermal conductivity of solid, BTU/hr-ft-degF
      alpha    thermal diffusivity of solid, sq ft/hr
      dikeArea[2] dike area [0] = floor area, sq ft
                            [1] = wall area                          */

#include <conio.h>
#include <math.h>
#include <stdio.h>

/*  function prototypes                                              */

double erf (double);
void oneLayerBoiloff (double, double, double, double, double [], double
                      *, double *);
int testVaporVol (double, double, double, double, double *, double *);

void oneLayerBoiloff (double t0, double k, double alpha,double dikeVaporVol,
 double dikeArea [], double *timeod, double *dispST)
{                                                                     */
int odflag = 0;          /* flag=1 when vapor overflows dike         */
double boiloffRate,      /* rate of evap of LNG from entire dike, #/hr */
       cumeFlux,         /* Q/A, BTU/sqft at any time                */
       effDikeArea,      /* dike area available for heat transfer/latent
                            heat evaporation of LNG, # sqft/BTU      */
       heatFlux,         /* q/A, BTU/hr-sqft                         */
       latHeat = 220.0,  /* latent heat evap LNG                     */
       time=0.0,         /* seconds                                  */
       timeHrs,          /* hours
          timeTable[]={0.001,0.1,0.5,1.0,2.0,5.0,10.0,25.0,50.0,100.0,250.0,500.
                       1000.0,1500.0,1800.0,2100.0,2400.0,2700.0},/*time
for print,seconds
*/        t,                /* temperature at x, degF                 */
       term,
       ts = -260.0,      /* boiling point of LNG, degF               */
       vaporWt = 0.0,    /*total weight of LNG vaporized, lbs        */
       x = 0.005;        /* distance in from surface, inches         */

int i=0;
term = x / 24.0;          /* 24 = 2 * 12          units conversion    */
effDikeArea = (dikeArea[0] + dikeArea[1]) / latHeat;
while (time < 2700.0){
       timeHrs = (time = timeTable[i++]) / 3600.0;
       t = ts + (t0-ts) * erf (term / sqrt (alpha * timeHrs));
       heatFlux = k * (t0-ts) / sqrt (3.14156 * alpha * timeHrs);
       boiloffRate = heatFlux * effDikeArea;
       cumeFlux = 2.0 * k * (t0-ts) * sqrt (timeHrs / (3.14156 * alpha));
       if (time != 0.001){
              vaporWt = cumeFlux * effDikeArea;
              if(!odflag) odflag = testVaporVol(dikeVaporVol, time,
                                   vaporWt, boiloffRate, timeod, dispST);}

          if (time == 0.1 || time == 0.5){
          fprintf (stdprn, "%4.1f      %9.1f      %11.1f    %10.1f     %7.2f\n",
                                   time,heatFlux,boiloffRate,vaporWt,t);}

          else{
       fprintf (stdprn, "%4.0f      %9.1f      %11.1f    %10.1f     %7.2f\n",
                                   time,heatFlux,boiloffRate,vaporWt,t);}

       }
return;
}  /*  oneLayerBoiloff                    dike.lib     */
```

```
/****  04/27/89  boiloff from single homogeneous material    onelboff
      according to the formula t=ts+(t0-ts)*erf(X) where X=x/2t
      ARGUMENTS
      to      ambient temperature, deg F
      k       thermal conductivity of solid, BTU/hr-ft-degF
      alpha   thermal diffusivity of solid, sq ft/hr
      dikeArea[2] dike area [0] = floor area, sq ft
                            [1] = wall area                          */

#include <conio.h>
#include <math.h>
#include <stdio.h>

/*  function prototypes                                              */

double erf (double);
void oneLayerBoiloff (double, double, double, double, double [], double
                      *, double *);
int testVaporVol (double, double, double, double, double *, double *);

void oneLayerBoiloff (double t0, double k, double alpha,double dikeVaporVol,
 double dikeArea [], double *timeod, double *dispST)
{
int odflag = 0;        /* flag=1 when vapor overflows dike              */
double boilOffRate,    /* rate of evap of LNG from entire dike, #/hr    */
       cumeFlux,       /* Q/A, BTU/sqft at any time                     */
       effDikeArea,    /* dike area available for heat transfer/latent
                          heat evaporation of LNG, # sqft/BTU           */
       heatFlux,       /* q/A, BTU/hr-sqft                              */
       latHeat = 220.0, /* latent heat evap LNG                         */
       time=0.0,       /* seconds                                       */
       timeHrs,        /* hours                                         */
          timeTable[]={0.001,0.1,0.5,1.0,2.0,5.0,10.0,25.0,50.0,100.0,250.0,500.
                       1000.0,1500.0,1800.0,2100.0,2400.0,2700.0},/*time
for print,seconds
*/        t,              /* temperature at x, degF                      */
       term,
       ts = -260.0,    /* boiling point of LNG, degF                    */
       vaporWt = 0.0, /*total weight of LNG vaporized, lbs              */
       x = 0.005;      /* distance in from surface, inches              */

int i=0;
term = x / 24.0;        /* 24 = 2 * 12          units conversion        */
effDikeArea = (dikeArea[0] + dikeArea[1]) / latHeat;
while (time < 2700.0){
        timeHrs = (time = timeTable[i++]) / 3600.0;
        t = ts + (t0-ts) * erf (term / sqrt (alpha * timeHrs));
        heatFlux = k * (t0-ts) / sqrt (3.14156 * alpha * timeHrs);
        boilOffRate = heatFlux * effDikeArea;
        cumeFlux = 2.0 * k * (t0-ts) * sqrt (timeHrs / (3.14156 * alpha));
        if (time != 0.001){
                vaporWt = cumeFlux * effDikeArea;
                if(!odflag) odflag = testVaporVol(dikeVaporVol, time,
                                     vaporWt, boilOffRate, timeod, dispST);}
           if (time == 0.1 || time == 0.5){
           fprintf (stdprn, "%4.1f     %9.1f     %11.1f    %10.1f    %7.2f\n",
                                     time,heatFlux,boilOffRate,vaporWt,t);}
           else{
        fprintf (stdprn, "%4.0f     %9.1f     %11.1f    %10.1f    %7.2f\n",
                                     time,heatFlux,boilOffRate,vaporWt,t);}
        }
return;
}  /*  oneLayerBoiloff                  dike.lib    */
```

```
/****   09/06/89     prints press any key etc                    pakc        */

#include <conio.h>
#include <stdio.h>

/*  function prototypes                                                      */

void barText (int, char []);
void pakc (int, int);
void splash ( int, int, int far *, char[], int);

void pakc (int x, int y)
{
int key;
extern int far * videoptr;
splash (x, y, videoptr, "press any key to continue", 0x0700);
barText (30, "alt Q to abort  ");
key = getch ();
if (key == 0){
        key = getch();
        if (key == 16) exit (0);}
return;
}  /*  pakc          bgen.lib    */
```

```
/****  06/01/89  to calc flow from pipe in tank as function  pipeflow
                      of time in minutes

        ARGUMENTS:
        pipe_i_d,       inches
        htUllage,       height of ullage in tank, ft
        ullagePress     vapor pressure of ullage in tank, psig
        dikeArea,       dike floor area, sq ft
        tankArea        tank floor area, sq ft
        spillTime,      duration of spill, minutes
        spillRate,      initial spill rate, gpm                          */
        RETURN:         time to empty tank, minutes

#include <conio.h>
#include <math.h>
#include <stdio.h>
                                                                         */
/*  function prototypes

double pipeFlow (double, double, double, double, double, double, double *);
double pipeLeak (double, double, double);

double pipeFlow (double pipe_i_d, double htUllage, double ullagePress,
          double dikeArea, double tankArea, double spillTime, double *spillRate
{
double flow,      /* cuft/min   */
          time = 0.0,     /* min   */
          liqHtinDike = 0.0,  /*    ft  */
          pressHead;          /* ft head equivalent to the ullage pressure  */

pressHead = ullagePress / 0.191;
while ((htUllage+pressHead > liqHtinDike)&&(htUllage > 0.0)&&(time <=
                          spillTime)){
        time++;
        flow = pipeLeak (pipe_i_d, (htUllage - liqHtinDike), ullagePress) /
                                   7.48;
        if (time == 1.0) *spillRate = 7.48 * flow;
        htUllage -= flow / tankArea;
        liqHtinDike += flow / dikeArea;
/*   printf ("time =%3.0f, ht in tank=%.0f, ht in dike=%.0f, flow=%.0f\n",
                  time, htUllage, liqHtinDike, flow);     */
                                   }
/*printf ("time in minutes to empty tank = %f\n",time);*/
return (time);
}     /* pipeFlow                        dike.lib  */
```

```
/****  03/09/89                                                    pipeleak
      return leak rate gpm given pipe size and liq head based on
          AGA report April,78 appendix a, pages 67-68
          ARGUMENTS:
          pipe_i_d          pipe i.d., inches
          htUllage          height of liq in tank, ft
          ullagePress       vapor pressure of liquid in tank, psig
          RETURN            flow, gpm                                  */

#include <math.h>

/*  function prototypes                                                */

double pipeLeak (double, double, double);

double pipeLeak (double pipe_i_d, double htUllage, double ullagePress)
{
double coef,pratio;

pratio = (ullagePress + 14.7 + 0.191 * htUllage) / 14.7;/* 0.191=(rho=27.5)/144
coef = (pratio < 2.0) ? sqrt (pratio - 1.0) : sqrt (0.5 * pratio);
return 105.2 * coef * pipe_i_d * pipe_i_d;
/* 105.2=0.61*sqrt(64.4*rho*144*14.7)*60*7.48/rho*.786/144            */
}  /*  pipeLeak                          dike.lib    */
```

```
                                                                please      */
/****  03/01/89

#include <conio.h>
#include <stdio.h>
                                                                            */
/*  function prototypes

void please (int, int);

void please (int x, int y)
{
mycputs (x, y++, "Please input the values requested");
gotoxy (x, y);
return;
}    /* please                                  dike.lib  */
```

```
/*  06/01/89    to get user response to a question flashed    question
                on the screen. First response choice returns 1,
                second response choice returns 0. Alt-Q aborts                  *

#include <conio.h>;
#define reverseColor textattr(BLACK + (LIGHTGRAY<<4));
#define stdColor textattr(LIGHTGRAY + (BLACK<<4));

/*  function prototypes                                                         */

void drawBorder (int, int, int, int);
int question (int, int, char[], char[]);

int question (int x, int y, char response1[], char response2[])
{
int loc, x1, width;
char key, key2;

x1 = 3 + strlen (response1);
width = x1 + 2 + strlen (response2);
x1 += x;
cursorOff ();
drawBorder (x, y, width, 3);
gotoxy (x += 3, y += 1);
reverseColor;
cputs (response1);
loc = 0;
stdColor;
mycputs (x1, y, response2);

while ((key = getch ()) != '\r'){
        stdColor;
        if (key == 0){
                key2 = getch ();
                switch (key2){
                        case 16:                        /* abort  */
                        exit (0);

                        case 77:                        /* right arrow  */
                        if (loc) break;
                        loc = 1;
                        gotoxy (x, y);
                        cprintf (response1);
                        reverseColor;
                        gotoxy (x1, y);
                        cprintf (response2);
                        break;

                        case 75:                          /* left arrow  */
                        if (!loc) break;
                        loc = 0;
                        gotoxy (x1,y);
                        cprintf (response2);
                        gotoxy (x, y);
                        reverseColor;
                        cprintf (response1);
                        break;
                default: break;}
                }   }
 stdColor;
 cursorOn (0);
 return loc;
 }   /*  question                    bgen.lib  */
```

```
                                                                    report
/****  09/25/89
      to start the output report of the dike program                        */

#include <conio.h>
#include <stdio.h>
#include <time.h>

                                                                            */
/*  function prototypes

void report(int, int, int, double, double, double [], double, double,
            double, double [],double, double [][4], double, double, double);

void report(int shape, int style, int typecon,
     double tankDia, double tankHeight, double dimension[], double
     htUllage, double ullagePress, double ambientTemp, double thickness[],
     double windSpeed,double propts[][4],
         double soilMoisture, double tankVol, double dikeVol)

{
int now, typeconWall;
char *strnow,                    /* string holding date and time    */
          *styles[3] = {"straight sides          ",
                        "sloped sides            ",
                        "sloped sides/floor shelf"},
          *matl[5] = {"gunite    ",
                      "concrete  ",
                      "concrete  ",
                      "earth     ",
                      "loose rock"},
                weacon,words[]="gunite   concrete  loose-rock";

time(&now);
strnow=ctime(&now);
typeconWall = typecon;
if (typecon == 2) typeconWall = 0;
fputs ("\f                                GAS RESEARCH INSTITUTE\n"
                "                             LNG Spill Simulation Program\n"
        "\t\t\t\t\t\t   page 1 of 2 pages \n",stdprn);
fprintf (stdprn, "Date/Time:%s \n",strnow);
fputs ("----------DIKE----------------   -----------TANK---------------\n"
       ,stdprn);
fprintf (stdprn,"shape  %s                    Diameter        %.1f  ft\n",
        (shape==0?"circular   ":"rectangular"),tankDia);
fprintf (stdprn,"style  %s    Height          %.1f  ft\n",
                                                    styles[style],ta
                                                              "
if (style == 1) fprintf (stdprn, "angle from vertical %.1f
                          "Volume %.1f cu ft\n",dimension[3]* 57.296, tank
else fprintf (stdprn, "\t\t\t\t   Volume %.1f cu ft\n", tankVol);
fprintf (stdprn,"Height    %.1f\n", dimension[0]);
if (shape){
fprintf (stdprn,"Length   %.1f                          Liquid height    %.1f  ft\n"
        dimension[1], htUllage);
fprintf (stdprn,"Width    %.1f                          Vapor pressure   %.1f psig\n
                dimension[2], ullagePress);
fprintf (stdprn, "Volume %.1f cu ft\n", dikeVol);}
else{
fprintf (stdprn,"Diameter  %.1f                         Liquid height  %.1f  ft\n",
        dimension[1], htUllage);
fprintf (stdprn,"Volume %.1f cu ft                    Vapor pressure   %.1f  psig\n"
                , dikeVol, ullagePress);}

fputs ("Floor surface\n",stdprn);
fprintf (stdprn,"     material  %s              -------Weather Conditions-------\n"
           ,matl[typecon]);
fprintf (stdprn,"    thickness  %.2f inches         Ambient temperature  %.1f de
```

```
          thickness[0], ambientTemp);
fprintf (stdprn,"   insulation  %.2f inches          Wind velocity          %.1f m
           thickness[2],windSpeed / 1.46666667);
fprintf(stdprn,"Wall surface                          \n");
fprintf (stdprn, "     material  %s\n",matl[typeconWall]);
fprintf (stdprn, "    thickness  %.2f inches\n",thickness[1]);
fprintf (stdprn,"   insulation  %.2f inches\n\n",thickness[3]);
fputs ("----------------------Material Properties--------------------------"
         "------\n",stdprn);
fprintf(stdprn,
"                        soil  insulation  %s\n",words);
fprintf (stdprn,
          "density,#/cu ft          %.2f    %.2f    %.2f    %.2f     %.2f\n",
        propts[2][0], propts[4][0], propts[0][0], propts[1][0], propts[3][0]);
fprintf (stdprn,
         "heat capacity,BTU/#,F    %.2f     %.2f      %.2f      %.2f       %.2f\n
        propts[2][1], propts[4][1], propts[0][1], propts[1][1], propts[3][1]);
fprintf (stdprn,
         "thermal cond,BTU/hr,ft,F %.2f     %.2f      %.2f      %.2f       %.2f\n
        propts[2][2], propts[4][2], propts[0][2], propts[1][2], propts[3][2]);
fprintf (stdprn, "Moisture, #/# dry soil    %.2f\n\n", soilMoisture);
return;
}  /*  report                    dike.lib  */
```

```
/****  05/27/89  to add spill description to output report   report0     */

#include <stdio.h>

/*  function prototypes                                                   */

void report0 (int, double, double, double);

void report0 (int spillMode, double pipeid, double spillRate, double spillTime)
{
fputs ("- - - - - - - - - - - Spill Description - - - - - - - - - - - - - -\n",
          stdprn);
switch (spillMode){
        case 0:
                fprintf (stdprn, "The spill is assumed to be of sufficient"
                " size to cover the dike floor instantaneously but to a very"
                " shallow depth thus allowing most of the dike volume to"
                " accumulate the vapor formed.  Following the initial spill,"
                " the spill rate is the boil-off rate.");
                break;
        case 1:
                fprintf (stdprn, "The simulation which follows is for a spill
                " of %7.2f gallons per minute for a total time of %7.2f "
                "minutes, after which the leak has stopped but the boil-off"
                " may continue.",spillRate, spillTime);
                break;
        case 2:
                fprintf (stdprn, The spill is from a broken pipe %4.1f inches"
                " in diameter, and lasts for %7.2f minutes.  The initial spill"
                " rate from this pipe is %7.2f gallons per minute.",pipeid,
                spillTime, spillRate);
fputs ("\n- - - - - - - Spill Consequences (for dike as is) - - - - - - - - -"
          "\n\n",stdprn);
fputs ("--------------------Simulation Details----------------------\n\n
          ,stdprn);
fputs ("time  heat rate to LNG   boil off     lbs LNG   surface temp\n",stdprn);
fputs (" sec   BTU/hr/sq ft  ·  rate #/hr    vaporized       degF\n",stdprn);
return;
}   report0                              dike.lib  */
```

```
/****   04/30/89                                                report1
       to print the second part of the dike report                        */

#include <conio.h>
#include <stdio.h>

/* function prototypes                                                    */

void report1 (double);

void report1 (double time1OverDike)
{
if(time1OverDike == 0.0){
       fputs("\nFor the original dike, vapor overflow does not occur in\n"
                "the first 45 minutes after the spill starts.",stdprn);
       mycputs (1, 3, "For the original dike, 45 min passes");
       mycputs (1, 4, "without vapor overflow");}
else{
fputs ("\nThe LNG vapor cloud overflowed the uninsulated dike\n", stdprn);
mycputs (1, 1, "The LNG vapor cloud overflowed the uninsulated dike");
fprintf (stdprn, "%.1f seconds after spill started\n", time1OverDike);
gotoxy (1, 2);
printf ("%.1f seconds after spill started", time1OverDike);}
fputs ("\f                              GAS RESEARCH INSTITUTE\n"
       "                              LNG Spill Simulation Program\n"
         "\t\t\t\t\t\t   page 2 of 2 pages \n",stdprn);
fputs ("- - - - Spill Consequenses (for dike with insulation) - - - - -\n"
       ,stdprn);

fputs ("\n---------------------Simulation Details----------------------\n\n"

fputs("time  heat rate to LNG   boil off     lbs LNG   surface temp\n",stdprn);
fputs (" sec   BTU/hr/sq ft     rate #/hr    vaporized      degF\n",stdprn);
return;
}   /*  report1                dike.lib  */
```

```
                                                                report2
/****   04/30/89
       to print the third part of the dike report                        */

#include <conio.h>
#include <stdio.h>
                                                                          */
/* function prototypes

int report2 (double);

int report2 (double time2OverDike)
{
if(time2OverDike == 0.0){
        fputs("\nFor the insulated dike, vapor overflow does not occur in\n"
                "the first 45 minutes after the spill starts.",stdprn);
        mycputs (1, 3, "For the insulated dike, 45 min passes");
        mycputs (1, 4, "without vapor overflow");}
else{
fputs ("\nFor the insulated dike, vapor overflow would begin\n", stdprn);
mycputs (1, 3, "For the insulated dike, vapor overflow would begin");
fprintf (stdprn, "%5.1f seconds after spill started\n", time2OverDike);
gotoxy (1, 4);
printf ("%5.1f seconds after spill started", time2OverDike);}
mycputs (10, 12, "                                          ");
mycputs (1, 6, "Would you like the downwind dispersion distances ?");

return (question (10, 8, " yes ", " no "));
}   /* report2                              dike/lib  */
```

```
/****  09/25/89  to see if a rerun is reqd                    rerun      */

#include <conio.h>;
#define reverseColor textattr (BLACK + (LIGHTGRAY<<4));
#define stdColor textattr (LIGHTGRAY + (BLACK<<4));

/*  function prototypes                                                   */

void drawAbox (int, int, int, int, char *, char *);
int getKey (int, int, char *[], int, int);
int rerun (void);

int rerun (void)
{
int x = 19, y = 7;
char *choices[2] = {"run new case", "exit program"};
drawAbox (x, y, 18, 2, "Options", "Choose one");
return (getKey (x+2, y+3, choices, 2, 0));
}  /*  rerun                dike.lib   */
```

```
/*  09/04/89  question requiring a yes/no response                respond
                                                                          */
  function prototypes

void mycputs (int, int, char[]);
int question (int, int, char[], char[]);
int respond (int, int, char[]);

int respond (int x, int y, char query[])
{
mycputs (x, y, query);
return (question (x+8, y+1, " yes ", " no "));
}   /*  respond                              bgen  */
```

```
/****  03/10/89    function returns the value of sigy        sigyz
          and sigz (based on info in ADL program)
          ARGUMENTS:
          xlee    -  distance from dike, ft
          weather -  letter B thru F signifying Gifford atmospheric categories
          sigy    -  sigma y
          sigz    -  sigma z                                                  */

#include <math.h>

/*  function prototypes                                                       */

void sigyz (double, char, double *, double *);

void sigyz (double xlee, char weather, double *sigy, double *sigz)
{
double con, con1, con2, con3, power = 0.919, term;
switch (weather){
        case 'B':
                con = 158.0;
                con1 = 2.041;
                con2 = 1.048;
                con3 = 0.041;
                power = 0.9;
                break;
        case 'C':
                con = 104.0;
                con1 = 1.786;
                con2 = 0.914;
                con3 = 0.0;
                power = 0.913;
                break;
        case 'D':
                con = 69.0;
                con1 = 1.505;
                con2 = 0.737;
                con3 = -0.105;
                break;
        case 'E':
                con = 51.0;
                con1 = 1.332;
                con2 = 0.678;
                con3 = -0.112;
                break;
        case 'F':
                con = 34.0;
                con1 = 1.146;
                con2 = 0.65;
                con3 = -0.113;}
term = xlee / 3280.8;
*sigy = 3.2808 * con * pow (term, power);
term = log10 (term);
*sigz = 3.2808 * pow (10.0, con1 + term * (con2 + term * con3));
return;
}  /*  sigyz                              dike.lib  */
```

```
/****  08/10/89  to write general msg to part of screen      smalspla
        ARGUMENTS:
        x                x location of upper left corner of print area
        y                y location of upper left corner of print area
        videoptr  address of start of video ram
        msg[]          the actual message to be printed

#include <conio.h>
#include <stdio.h>
#define stdColor textattr (LIGHTGRAY + (BLACK<<4));


/* function prototypes                                                  */

void smallSplash (int, int, int far *, char[]);

void smallSplash (int x, int y, int far *videoptr, char msg[])
{
int far *videonow;
int far *videostart;
char c, *ch;
stdColor;
ch = msg;
videonow = videostart = videoptr + x + 80 * y;
while (*ch != ''){
        c = *ch++;
        switch (c){
                case '':
                        *(videonow++) = 250 | 0x0700;
                        break;
                case '':
                        *(videonow++) = 182 | 0x0700;
                        break;
                case '':
                        *(videonow++) = 199 | 0x0700;
                        break;
                case  '\n':
                        videonow = (videostart += 80);
                        break;
                case '':
                        *(videonow++) = 248 | 0x0700;
                        break;
                case '':
                        *(videonow++) = 218 | 0x0700;
                        break;
                case '':
                        *(videonow++) = 196 | 0x0700;
                        break;
                case '':
                        *(videonow++) = 179 | 0x0700;
                        break;
                case '':
                        *(videonow++) = 192 | 0x0700;
                        break;
                case '':
                        *(videonow++) = 217 | 0x0700;
                        break;
                case '':
                        *(videonow++) = 191 | 0x0700;
                        break;
                case '':
                        *(videonow++) = 197 | 0x0700;
                        break;
```

```
                case '':
                        *(videonow++) = 180 | 0x0700;
                        break;
                case '':
                        *(videonow++) = 193 | 0x0700;
                        break;
                case '':
                        *(videonow++) = 194 | 0x0700;
                        break;
                case '':
                        *(videonow++) = 195 | 0x0700;
                        break;
                case '':
                        *(videonow++) = 214 | 0x0700;
                        break;
                case '':
                        *(videonow++) = 211 | 0x0700;
                        break;
                case '':
                        *(videonow++) = 201 | 0x0700;
                        break;
                case '':
                        *(videonow++) = 205 | 0x0700;
                        break;
                case '':
                        *(videonow++) = 186 | 0x0700;
                        break;
                case '':
                        *(videonow++) = 200 | 0x0700;
                        break;
                case '':
                        *(videonow++) = 187 | 0x0700;
                        break;
                case '':
                        *(videonow++) = 188 | 0x0700;
                        break;
                case '':
                        *(videonow++) = 169 | 0x0700;
                        break;
                case '':
                        *(videonow++) = 170 | 0x0700;
                        break;
                case '':
                        *(videonow++) = 208 | 0x0700;
                        break;
                case '':
                        *(videonow++) = 219 | 0x0700;
                        break;
                case '':
                        *(videonow++) = 222 | 0x0700;
                        break;
                case '':
                        *(videonow++) = 229 | 0x0700;
                        break;
                case '':
                        *(videonow++) = 233 | 0x0700;
                        break;
                case '':
                        *(videonow++) = 215 | 0x0700;
                        break;
                default:
                        *(videonow++) = c | 0x0700;}}
return;
```

```
/****  05/23/89  to write general msg to video ram              splash      */

#include <conio.h>
#include <stdio.h>

/*  function prototypes                                                      */

void splash (int x, int y, int far *videoptr, char msg[], int colr)
{
int far *videonow;
int far *videostart;
char c;
char *ch;

ch = msg;
videonow = videostart = videoptr+x-1+80*(y-1);
while (*ch != ''){
        c = *ch++;
        switch (c){
                case  '\n':
                        if(videonow - videostart < 160)
                                videonow = (videostart += 80);
                        break;
                case '':
                        *(videonow++) = 218 | colr;
                        break;
                case '':
                        *(videonow++) = 196 | colr;
                        break;
                case '':
                        *(videonow++) = 179 | colr;
                        break;
                case '':
                        *(videonow++) = 192 | colr;
                        break;
                case '':
                        *(videonow++) = 217 | colr;
                        break;
                case '':
                        *(videonow++) = 191 | colr;
                        break;
                case '':
                        *(videonow++) = 197 | colr;
                        break;
                case '':
                        *(videonow++) = 180 | colr;
                        break;
                case '':
                        *(videonow++) = 193 | colr;
                        break;
                case '':
                        *(videonow++) = 194 | colr;
                        break;
                case '':
                        *(videonow++) = 195 | colr;
                        break;
                case '':
                        *(videonow++) = 214 | colr;
                        break;
                case '':
                        *(videonow++) = 211 | colr;
                        break;
                case '':
```

```
                        *(videonow++) = 201 | colr;
                        break;
                case '':
                        *(videonow++) = 205 | colr;
                        break;
                case '':
                        *(videonow++) = 186 | colr;
                        break;
                case '':
                        *(videonow++) = 204 | colr;
                        break;
                case '':
                        *(videonow++) = 185 | colr;
                        break;
                case '':
                        *(videonow++) = 200 | colr;
                        break;
                case '':
                        *(videonow++) = 187 | colr;
                        break;
                case '':
                        *(videonow++) = 188 | colr;
                        break;
                case '':
                        *(videonow++) = 169 | colr;
                        break;
                case '':
                        *(videonow++) = 170 | colr;
                        break;
                case '':
                        *(videonow++) = 208 | colr;
                        break;
        case '':
                        *(videonow++) = 215 | colr;
                        break;
                case '':
                        *(videonow++) = 219 | colr;
                        break;
                case '':
                        *(videonow++) = 222 | colr;
                        break;
                case '':
                        *(videonow++) = 249 | colr;
                        break;
                default:
                        *(videonow++) = c | colr;}}
return;
}  /*  splash                bgen.lib    */
```

```
/****  06/23/89 to write logo screen and disclaimer          splash0
                    directly to videoram

#include <conio.h>
#include <dos.h>
#include <stdio.h>

/*  function prototypes                                              */

void baseLine (void);
void cursorOff(void);
void splash0 (int, int, int far *, int);
void pakc (int, int, int far *, int);
void splash (int, int, int far *, char[], int);

void splash0 (int x, int y, int far *videoptr, int colr)
{
char words[] = "                                        "
                "                                              \n"
                ""
                "\n"
                "                                                 "
                "                                              \n"
                ""
                "\n"
                "                                                 "
                "                                              \n"
                ""
                "\n"
                "                                                 "
                "                                              \n"
                "                          W E L L B O R N  S Y S T E M "
                "S                         \n"
                "                                                 "
                "                                              \n"
                "                                                 a  "
                "                                            \n"
                "                                                 "
                "                                              \n"
                "                                 d i v i s i o n  "
                "  o f                             \n"
                "                                                 "
                "                                              \n"
                "                       R o b e r t  F .  B e n"
                " e n a t i  I n c .                              \n"
                "                                                 "
                "                                              \n"
                "                              7 5  D e e p d a l e"
                "   D r i v e                                   \n"
                "                                                 "
                "                                              \n"
                "                        M a n h a s s e t "
                "    N . Y .  1 1 0 3 0                           \n"
                "                                                 "
                "                                              \n"
                "                                  516 Un 9 8440"
                "                                    \n"
                "                                                 "
                "                                              \n"
                "                                                 "
                "                                              \n"
                ""
                ""
```

```
                                "                                       "
                                "                                   ";

char msg [] = "WELLBORN SYSTEMS DISCLAIMER\n\n\nLEGAL NOTICE  Wellborn"
            " Systems warrants that this program will substantially\nconform"
            " to the specifications described in the documentation provided\n"
            "it is used on the computer hardware and with the operating\n"
            "system for which it was designed.\n \nExcept as specifically"
            " provided above, Wellborn Systems makes no warranty or\nrepresent"
            "ation, either expressed or implied, with respect to this program"
            " or\ndocumentation, including their quality, performance, merchan"
            "tability, or\nfitness for a particular purpose.\n\nBecause computer"
            " programs are inherently complex and may not be completely\nfree "
            "of errors, you are advised to verify your work.  In no event will"
            "\nWellborn Systems be liable for direct, indirect, special, incid"
            "ental, or\nconsequential damages arising out of the use or inabil"
            "ity to use the\nprogram or documentation, even if advised of the "
            "possibility of such\ndamages.  In particular, Wellborn Systems is"
            " not responsible for any costs\nincludind but not limited to those"
            " incurred as a result of lost profits\nor revenue, loss of use of"
            " the computer program, loss of data, the cost\nof a substitute pro"
            "gram, claims by third parties or for other similar costs.";

clrscr ();
cursorOff ();
splash (x, y, videoptr, words, colr);
sleep (3);
clrscr ();
splash (x, y, videoptr, msg, colr);

splash (1, 25, videoptr, "  Wellborn Systems                                "
                "                       copyright 1989   ", 0x7000);
pakc (24, 24, videoptr, colr);
window (1, 1, 80, 24);
clrscr ();
return;
}  /* splash0                bgen.lib  */
```

```
/****     09/07/89  to write opening screen                    splash1   */

#include <conio.h>
#include <stdio.h>
                                                                        */
/* function prototypes

void pakc (int, int, int far *, int);
void splash (int, int, int far *, char *, int);
void splash1 (int far *, int);


void splash1 (int far *videoptr, int colr)
{
char msg[] = "                                  GAS RESEARCH INSTITUTE\n"
                                                      LNG Spills\n"
                "                                  Vaporization and Dispersion\n\n"
                "                                  "This program is intended for evaluation of the"
                " effectiveness of LNG storage\ndike insulation material to"
                " aid decision making regarding installation of new\ndike"
                " insulation. Depending on several site-specific conditions"
                " and candidate\nmaterials, dike insulation can significantly
                " mitigate hazards associated with\naccidental LNG spills"
                " within storage dikes by minimizing vaporization rates and\n
                "downwind dispersion.  The program permits evaluation of"
                " dike insulation for\nboth uninsulated dikes and insulated"
                " dikes considered for insulation retrofit.\n\nUser-specified
                " input regarding a storage tank and its surrounding diked"
                " area\nis used to calculate vaporization rates for a user-"
                "defined spill scenario.\nCalculated boil-off rates are"
                " provided for cases with and without the addition\nof new"
                " dike insulation. Next, the user is provided the option of"
                " calculating\nvapor dispersion distances for concentrations"
                " of interest using a simple\nGaussian dispersion algorithm."
                "Dispersion calculations are provided for\ncomparative purpos
                " only as use of this algorithm for safety evaluation or\n"
                "regulatory compliance is not recommended by GRI.";
 splash (1, 1, videoptr, msg, colr);
 pakc (24, 24, videoptr, colr);
 window (1, 1, 80, 24);
 clrscr ();
 return;
 }   /*  splash1                    dike.lib   */
```

```
/***   09/13/89                                                    tank
     to request tank and liquid dimensions and return tank volume        */

#include <conio.h>
#include <dos.h>
#include <math.h>
#include <stdio.h>
#define barColor textattr (BLACK + (LIGHTGRAY<<4));
#define stdColor textattr (LIGHTGRAY + (BLACK<<4));

/* function prototypes                                                   */

void cleanSpace (int, int, int);
void cursorOn (int);
double tank (double *, double *, double *, double *, double *, double *);
double getNum (void);



double tank (double *tankHeight, double *tankDia, double *htUllage,
             double *ullagePress, double *ullageVol, double *tankArea)
{
window (1, 1, 80, 25);
barColor;
mycputs (30, 25, "   numerics only                ");
stdColor;
mycputs (15, 2, "Tank and Liquid Details");
gotoxy (5, 3);
please (10, 22);
mycputs (5, 5, "Diameter, ft = ?");
mycputs (5, 7, "Height, ft = ?");
mycputs (5, 9, "Height of liquid in tank, ft = ?");
mycputs (5, 11, "LNG vapor pressure, psig = ?");
gotoxy (20, 5);
cursorOn(0);
*tankDia = getNum ();
if (*tankDia == 0.0){
        mycputs( 70, 3, "null");
        *tankHeight = 0.0;
        *htUllage = 0.0;
                *ullagePress = 0.0;
        goto out;}
gotoxy (70, 3);
cprintf ("%.1f", *tankDia);
gotoxy (18, 7);
*tankHeight = getNum();
gotoxy (70, 4);
cprintf ("%.1f", *tankHeight);
redo:gotoxy (36, 9);
*htUllage = getNum ();
if (*htUllage > *tankHeight && error (1)){
        cleanSpace (35, 9, 9);
        goto redo;}
gotoxy (70, 5);
cprintf ("%.1f", *htUllage);
gotoxy (32, 11);
*ullagePress = getNum ();
gotoxy (70, 6);
cprintf ("%.2f", *ullagePress);
out:*ullageVol = *htUllage * (*tankArea = 0.7854 * *tankDia * *tankDia);
window (1, 1, 52, 24);
clrscr();
window (1, 1, 80, 25);
```

```
return (*tankHeight * *tankArea);
}   /*  tank                                    dike.lib  */
```

```
/****  04/16/89                                                    text
      to print text related to input screens                              */

/*  function prototypes                                                   */

void text (int);

void text (int line)
{
int x = 2;
gotoxy (x, 16);
switch (line){
        case 0:
        cputs ("This condition assumes a spill of sufficient size  ");
        mycputs (x, 17, "to cover the dike floor instantaneously but to a  ");
        mycputs (x, 18, "very shallow depth thus allowing most of the dike ");
        mycputs (x, 19, "volume to accumulate the vapor formed.Following the");
        mycputs (x, 20, "initial spill, the spill rate is the boil-off rate ");
        mycputs (x, 21, "          ");
        break;
        case 1:
        cputs ("This condition takes a specified spill rate and    ");
        mycputs (x, 17, "simulates its spread over the dike floor and the   ");
        mycputs (x, 18, "subsequent vaporization which results therefrom.   ");
        lastline:
        mycputs (x, 19, "Since a specified spill time is involved, the      ");
        mycputs (x, 20, "vapor cloud is of finite size and may not overflow ");
        mycputs (x, 21, "the dike.");
        break;
        case 2:
        cputs ("This condition assumes a ruptured pipe of specified");
        mycputs (x, 17, "size. The leak rate from the ruptured pipe is      ");
        mycputs (x, 18, "calculated and becomes the LNG spill rate.         ");
        goto lastline;
        }
return;
}   /*  text                          dike.lib   */
```

```
/****  03/27/89  to control transient conduction calcs        transient
          ARGUMENTS:
          alpha[]    - thermal diffusivity earth/liner/insulation
          dikeArea[] - floor/walls, sq ft
          k[]        - thermal conductivity earth/liner/insulation,eng units
          T[]        - temperature at nodes
          x[]        - node lengths, ft
          nc[]       - nodes per zone
          n          - total node count                                    */
          nodePos[]  - node # at zone boundaries

#include <stdio.h>

/*  function prototypes                                                   */

void conduction (double, double [], double [], double, double [],
                                 double [], int [], int);
void transient (double [], double [], double [], double [], double [],
                                int [], int, int []);

void transient (double alpha[], double dikeArea[], double k[],
                                double T[], double x[], int nc[], int n, int nod

{
static double
          boilOffRate,  /* rate at which LNG boils off from entire dike, #/hr */
          deltime,                    /* time step, sec                       */
          latHeat = 220.0,            /* latent heat of evaporation of LNG    */
          prntInterval [] =
                {1.0,4.0,5.0,20.0,30.0,50.0,100.0,250.0},  /* seconds
          prntTime [] =
                {1.0, 5.0, 10.0, 30.0, 150.0, 500.0, 1000.0, 1.0e10}, /* seconds
                                      /* avg heat flow from surface to LNG     */
          q,                          /* rate of heat flow to surface,BTU/hr/sqft */
          qc = 0.0,                   /* qc at beginning of time step          */
          qold = 0.0,                 /* total BTU/sqft at any time            */
          sumq = 0.0,           /* simulation time at which printing occurred  */
          timeold = 0.0,
          ts = -260.0,                      /* simulation time, sec            */
          ytime,                            /* total weight of LNG vaporized, lbs */
          vaporWt;

 int i = 0;
 ytime = 0.0;
 while (ytime <= 391.0){
          if (ytime < 0.1){ deltime = 0.01; goto run;}
          if (ytime < 1.0){ deltime = 0.015; goto run;}
          deltime = 0.0225;
          run: qc = 2.0 * k[0] * (T[0] - ts) / x[0];
                    q = 0.5 * (qc + qold);
                    qold = qc;
                    /*sumq += q * deltime / 3600.0;*/
                    boilOffRate = q * dikeArea[0] / latHeat;
                    vaporWt += boilOffRate * deltime / 3600.0;
                    conduction (ts, alpha, k, deltime/3600.0, x, T, nc, n);
                    if (ytime == 0.0) || (ytime - timeold >= prntInterval[i])){
                          if (ytime >= prntTime[i]) i++;
                          goto prnt;}
                    else goto noprnt;
                    prnt: fprintf (stdprn, "%4.0f       %9.1f      %9.1f     %8.2f
                                      ytime, q, boilOffRate, vaporWt, T[0]);
                    timeold = ytime;
                    noprnt: ytime += deltime;}

 return;
 }   /*  transient                          dike.lib  */
```

```
                                                                twolayer
/****  06/02/89
      to calculate the near surface temperature, the heat flux, the
      boil off rate and the amount evaporated from
      a dike composed of two lawyers of different materials based on
      dt/dx=-(t0-ts)/[1+2sum^nexp(n^2l^2/  see Carslaw & Jaeger p322
      ARGUMENTS: t0  ambient temperature, degF
                        k[] thermal conductivity [0]=top layer on floor
                                                              [1]=top layer
                                                              [2]=bottom la
                        alpha[] thermal diffusivity [0]=top layer floor, sqft
                                                                    [1]=top
                                                                    [2]=bot

                  dikeArea[] [0]=floor area, sqft
                             [1]=wall area
                        l[] effective thickness of top layer [0]=on floor, in

#include <conio.h>
#include <math.h>
#include <stdio.h>
                                                                        */
/*  function prototypes

double erfc (double);
double sumFunction (double, double, double *);
double tempFunction (double, double, double, double);
int testVaporVol (double, double, double, double, double *, double *);
void twoLayerBoilOff (double, double [], double [], double, double [],
                                    double [], double *, double *);

void twoLayerBoilOff (double t0, double k[], double alpha[], double
dikeVaporVol, double dikeArea[], double l[], double *timeod, double *dispST)
{
int i = 0,
    odflag = 0;   /* flag = 1 when vapor overflows dike                  */

double amtEvap = 0.0,/* amount evaporated,#/hr from total dike           */
          beta,          /* beta = (sigma - 1)/(sigma + 1)                */
          betawall,      /* same as beta but for wall surface             */
          boilOffRate,   /* rate of evaporation, #/hr from entire dike     */
          dtolatht,      /* (t0-ts)/latent heat                           */
          effDikeArea,   /* dike area available for heat transfer         */
          heatFlux,      /* average q/A, BTU/hr-sqft                      */
          latHeat = 220.0, /* latent heat evaporation of LNG, BTU/#       */
          m1, mr1, prefix, prefix1, sqraot,
          term, term1,                                                    */
          time = 0.0,    /* seconds                                       */
          timeHrs,       /* hours
          timeTable[]={0.001,0.1,0.5,1.0,2.0,5.0,10.0,25.0,50.0,100.0,250.0,500.
               1000.0,1500.0,1800.0,2100.0,2400.0,2700.0},/*time for printout *
          t,             /* temperature at x, degF                        */
          ts = -260.0;  /* atmospheric boiling point of LNG, degF         */

 dtolatht = (t0 - ts) / latHeat;
 effDikeArea = dikeArea[0] + dikeArea[1];
 term = sqrt (alpha[1] / alpha[2]);          /* calculate gamma for walls  */
 t = term * k[2] / k[1];                     /* calculate sigma for walls  */
 betawall = (t - 1.0) / (t + 1.0);           /* calculate betawall         */
 term = sqrt (alpha[0] / alpha[2]);          /* calculate gamma for floor  */
 t = term * k[2] / k[0];                     /* calculate sigma for floor  */
 beta = (t - 1.0) / (t + 1.0);               /* calculate beta             */
 l[0] /= 12.0; l[1] /= 12.0;                 /* convert thicknesses to ft  */
 while (time < 2700.0){
        timeHrs = (time = timeTable[i++]) / 3600.0;
```

```
        sqraot = sqrt (alpha[0] / timeHrs);
        term = alpha[0] * timeHrs;
        t = tempFunction (beta, term, l[0], t0);/*calc temp near surface    */
        terml = l[0] * l[0] / term;
        prefix1 = dtolatht * sqraot * dikeArea[0];
        prefix = 2.0 * prefix1 * timeHrs;
        boilOffRate = sumFunction (beta, terml, &m1) * prefix1;/* floor     */
        amtEvap = m1 * prefix;
        term = alpha[1] * timeHrs;
        terml = l[1] * l[1] / term;
        prefix1 *= (dikeArea[1] / dikeArea[0]);
        prefix *= (dikeArea[1] / dikeArea[0]);
        boilOffRate += (sumFunction (betawall, terml, &m1) * prefix1); /*walls
        amtEvap += (m1 * prefix);
        heatFlux = boilOffRate * latHeat / effDikeArea;
        if (time == 0.001) amtEvap = 0.0;
        if (!odflag) odflag = testVaporVol (dikeVaporVol, time, amtEvap,
                                                              boilOffRate, ti
        if (time == 0.1 || time == 0.5){
        fprintf (stdprn, "%4.1f     %9.1f     %10.1f    %10.1f    %7.2f\n",
                    time, heatFlux, boilOffRate, amtEvap, t);}
        else{
        fprintf (stdprn, "%4.0f     %9.1f     %10.1f    %10.1f    %7.2f\n",
                    time, heatFlux, boilOffRate, amtEvap, t);}}
}

double sumFunction (double beta, double terml, double *amtEvap)
{
double betan,               /* beta^n                                      */
         m1, mr1, n, z, zs;

for (n = 1.0, betan = 1.0, m1=mr1=0.0; n <= 10.0; n++){
        if ((zs = n * n * terml) < 6.76){
                z = 1.0 / exp (zs);
                mr1 += (betan *= beta) * z;        /* 1.7725 = sqrt (pi)      */
        m1 += betan * (z - 1.7725 * sqrt (zs) * erfc (sqrt (zs)));}}
*amtEvap = 1.0 + 2.0 * m1;
return (1.0 + 2.0 * mr1);
}     /*  sumFunction  */

double tempFunction (double alpha, double term, double l, double t0)
{
double alphan = 1.0,       /* alpha^n                                      */
         n,
         sum = 0.0,
         ts = -260.0,
         t,
         x;

x = 0.000417 - l;
term = 0.5 / sqrt (term);
for (n = 0.0; n <= 5.0; n++){
        sum += alphan * (erfc (((n + n + 1.0) * l + x) * term) - alpha *
                                  erfc (((n + n + 1.0) * l - x) * term));
        alphan *= alpha;}
return t0 + (ts - t0) * sum;
}   /*  temp Function            dike.lib   */
```

```
/****  04/29/89  to test vapor vol relative to dike vol      vaporvol
          ARGUMENTS:
          dikeVapVol     - dike (less tank) vapor volume, cu ft
          time           - time, seconds
          vaporWt        - total wt vapor generated since time=0, #
          boilOffRate    - rate of vapor generation at time, #/hr
          timeOverDike   - time when vapor first overflowed dike, sec
          dispSourceTerm - rate of overflow at time, #/hr
                (timeOverDike & dispSourceTerm determined by linear
                 interpolation of two time spots)

/*  function prototypes                                              */

int testVoporVol (double, double, double, double, double *, double *);

int testVaporVol (double dikeVaporVol, double time, double vaporWt, double
                  boilOffRate, double *timeOverDike, double *dispSourceTerm)
{
double factor,
          vaporVol;
static double vaporVol1,
                           time1,
                           source1;
vaporVol = 9.121 * vaporWt;     /* 9.121=359*(460+-260)/492/16          */
if (vaporVol < dikeVapVol){
        vaporVol1 = vaporVol;
        time1 = time;
        source1 = boilOffRate;
        return 0;}
factor = (dikeVapVol - vaporVol1) / (vaporVol - vaporVol1);
*timeOverDike = time1 + factor * (time - time1);
*dispSourceTerm = source1 + factor * (boilOffRate - source1);
return 1;
}  /*  testVaporVol                          dike.lib  */
```

```
/****   04/07/89                                                warning
       to flash warning on screen and await response                        */

#include <conio.h>
#include <stdio.h>
#define brightColor textattr (WHITE + (BLACK<<4));
#define stdColor textattr (LICHTGRAY + (BLACK<<4));

/*  function prototypes                                                     */

void bleep (void);
void pakc (int, int);
int question (int, int, char[], char[]);
int warning (int, double, double);

int warning (int no, double dikeVol, double tankVol)
{
int action;
char buffer[60*7*2];
bleep();
gettext (1, 18, 52, 24, buffer);
window (1, 18, 52, 24);
clrscr ();
switch (no){
        case 0:
            gotoxy (1, 1);
            printf ("Dike volume (%.1f cuft) is less than",dikeVol);
            gotoxy (3,2);
            printf ("110%% of tank volume (%.1f cuft)",tankVol);
            pak:pakc (8, 5);
            puttext (1, 18, 52, 24, buffer);
                window (1, 1, 52, 24);
                clrscr ();
                break;
        case 1:
                brightColor;
                mycputs (12, 1, "Serious Warning");
            stdColor;
            gotoxy (1, 2);
            printf ("The dike volume (%.1f cuft) is inadequate to",dikeVol);
            gotoxy (1,3);
            printf ("contain the total tank volume (%.1f cuft)",tankVol);
            goto pak;
        case 2:
            mycputs (2, 1, "You have entered 0.0");
            mycputs (2, 2, "an improbable value!");
            act:action = question (2, 3, " accept ", " retry ");
            puttext (1, 18, 52, 24, buffer);
            window (1, 1, 80, 24);
            return action;}
window (1, 1, 80, 24);
return (0);
}   /*  warning                          dike.lib    */
```

```
                                                                weather
/****   09/25/89                                                          */
       to get weather category for transient calcs

#include <stdio.h>

                                                                          */
/*  function prototypes

void barText (int, char *);
void cursorOff (void);
void cursorOn (int);
void drawAbox (int, int, int, int, char *, char *);
int getKey (int, int, char *[], int, int);
char weather (void);

char weather ()
{
char line[53]="-----------------------------------------------------\0";
char weacon;
char letter[] = {'B','C','D','E','F'};
char *categories[5] = { " B moderately unstable",
                        " C slightly unstable",
                        " D neutral",
                        " E slightly stable",
                        " F moderately stable"};

int i;
barText (30, "selection keys only");
window (1, 1, 52, 24);
clrscr();
cputs("To calculate downwind dispersion, you must specify");
mycputs(1, 2, "the weather conditions by choosing from categories");
mycputs (1, 3, "B - F. See table below of meteorological categories");
mycputs (1, 4, line);
mycputs (1, 5, "surface");
mycputs (32, 5, "nighttime conditions");
mycputs (2, 6, "wind");
mycputs (9, 6, "daytime insolation");
mycputs (34, 6, "(amount overcast)");
mycputs (2, 7, "speed");
line[45] = '\0';
line[22] = line[23] = ' ';
mycputs (8, 7, line);
mycputs (2, 8, "mi/hr");
mycputs (8, 8, "strong moderate slight");
gotoxy (34, 8);
putch ('\xf2');
cputs ("1/2 ");
gotoxy (44, 8);
putch ('\xf3');
cputs ("3/8");
mycputs (2, 9, "< 4.5    A      A-B      B");
mycputs (2, 10, " 4.5    A-B      B       C        E         F");
mycputs (2, 11, "  9      B      B-C      C        D         E");
mycputs (2, 12, " 13.5    C      C-D      D        D         D");
mycputs (2, 13, "> 13.5   C       D       D        D         D");
cursorOff ();
drawAbox (16, 14, 26, 5, "Weather Conditions", "Select category");
weacon = letter[getKey (19, 17, categories, 5, 0)];
clrscr ();
window (1, 1, 80, 24);
gotoxy (69, 21);
putch (weacon);
return weacon;
}   /*  weather                                    dike.lib  */
```

```
/****   04/30/89   to get wind speed and ambient temperature,  windtemp
     and to put ambient temp into tempture vector for transient calcs    */

#include <stdio.h>
                                                                          */
/*  function prototypes

void cursorOff (void);
void cursorOn (int);
double getNum (void);
void windTemp (double *, double []);

void windTemp (double *windSpeed, double tempture [])
{
double ambientTemp;
int i;

window (1, 1, 52, 24);
clrscr();
window (1, 1, 80, 24);
mycputs (15, 1, "Weather conditions");
mycputs (5, 5, "wind speed, miles/hr = ?");
mycputs (5, 7, "ambient temperature, degF = ?");
cursorOn (0);
gotoxy (28,5);
*windSpeed = getNum ();
gotoxy (72, 22);
cprintf ("%.1f",*windSpeed);
*windSpeed *= 1.466666667;          /*1.46667 = 5280/3600   */
gotoxy (33,7);
ambientTemp = getNum ();
for (i = 0; i < 50; i++){
        tempture[i] = ambientTemp;}
gotoxy (71,21);
cprintf ("%.1f",ambientTemp);
window (1, 1, 52, 24);
clrscr();
return;
}   /*  windTemp                          dike.lib  */
```

```
/**** 10/08/89    to write scrollable text to a window        writetxt
        ARGUMENTS:
        ptext          pointer to text to be written
        e
        x,y            screen coordinates of ul corner of window
        w              width of window
        h              height of window
        color1         color attribute of main portion of text
        color2         color attribute of accented text

  function prototypes                                                   */

writText (char *, int *, int, int, int, int, int, int);

#include <global.h>

writText(char *ptext, int *e, int x, int y, int w, int h, int color1, int color2

{
register int i = 0, j = 0;
int atend = FALSE;
int color = color1;
int far *vP;
int len = 0;
int m = 0;
int more = TRUE;
vP = OA(x,y++);

if ((w == NULL) && (h == NULL)){
        w = strlen (ptext);
        h = 1;}                     /*   end if   */
else { w -= 4; h -= 2; }

while ((j < h) && (atend == FALSE)){
        *e += (len = getLen (ptext + m, w));
        for (i = 0; i < len; i++){
                switch (*(ptext + m++)){
                        case '' : color = color2; break;
                        case '' : color = color1; break;
                        case '' : color = color2; break;
                        case '' : color = color1; break;
                        case '' : *(vP++) = 174 | color; break;
                        case '' : *(vP++) = 175 | color; break;
                        case '' : *(vP++) = 176 | color; break;
                        case '' : *(vP++) = 177 | color; break;
                        case '' : *(vP++) = 178 | color; break;
                        case '' : *(vP++) = 192 | color; break;
                        case '' : *(vP++) = 194 | color; break;
                        case '' : *(vP++) = 196 | color; break;
                        case '' : *(vP++) = 205 | color; break;
                        case '' : *(vP++) = 217 | color; break;
                        case '' : *(vP++) = 219 | color; break;
                        case '' : *(vP++) = 220 | color; break;
                        case '' : *(vP++) = 223 | color; break;
                        case '' : *(vP++) = 248 | color; break;
                        case '' : *(vP++) = 249 | color; break;
                        case '' : *(vP++) = 254 | color; break;
                        case '\n' :
                        case '\r' : break;
                        case '' : more = FALSE; atend = TRUE; continue;
                        default  : *(vP++) = *(ptext + m - 1) | color; break;
                                }          /*   end switch   */
                        }                  /*   end for   */
```

```
        vP=OA(x,y++);
        j++;}
/*        *e=cumu+*e;*/
        return (more);
} /* writText
```

```
/*   end while   */

bgen.lib    */
```

```
#include <string.h>
#include <keys.h>
#include <global.h>

void drawClear(int, int, int, int, int, int);

writeText(char txt[],int col,int row,int wid,int hit,int c1,int c2)
{
int i;
int size;
int more=FALSE;
int str=0, end=0, tmpend=0;
int maxlines=0;
int adjwid=wid-4;
int line=0, ret;
char *pbuf;

if ((strchr(txt,233)) == NULL) strcat(txt, "");
size = strlen (txt);
pbuf = txt;
more = writText (pbuf, &end, col+2, row+1, wid, hit, c1, c2);
tmpend = end;
while (tmpend < size) {
        tmpend += getLen (pbuf+tmpend, adjwid);
        maxlines ++;
}
while ( (ret = getch()) != ESC ) {
        if (ret == 0) {
                ret = getch();
                switch (ret) {
                        case END :
                                if (more == FALSE) {
                                        bleep();
                                        break;}                 /*    end if    */
                                if (line <= (maxlines-(hit-2))) {       /* full
                                        while (end < size) {  /* set vars to las
                                                str += getLen (pbuf+str, adjwid)
                                                tmpend = end += getLen (pbuf+end
                                                line ++;}         /*   end while
                                        drawClear (col+1, row+1, wid-2, hit-2, c
                                        more = writText (pbuf+str, &tmpend, col+
                                                                wid, hit, c1,
                                        break;}                 /*   end if  */
                                else {                           /* partial rewrit
                                        int nrow = ((row+hit-1) - (maxlines-line
                                        tmpend=end;
                                        movetext (col+1, row+1+(maxlines-line),
                                                        row+hit-2, col+1, row+1)
                                        drawClear (col+1, row+(hit-(maxlines-lin
                                                        wid-2, maxlines-line, c
                                        while (end < size) {
                                                str+ = getLen (pbuf+str, adjwid)
                                                end += getLen (pbuf+end, adjwid)
                                                line ++;}         /*   end while
                                        more = writText (pbuf+tmpend, &tmpend, c
                                                                wid, 2+line, c
                                        break;}                 /*   end else  */
                        case PGUP :
                                if (line == 0){
                                        bleep();
                                        break;}                  /*   end if  */
                                if (line >= hit-2){               /* full rewrite
                                        tmpend = end;
```

```
                line -= hit-3;
                for (i=0, str=0; i<line; i++) str+=getLe
                drawClear (col+1, row+1, wid-2, hit-2, c
                more = writText (pbuf+str, &tmpend, col+
                                                wid, hit, c1,
                for (i=0, end=0; i<(hit-2)+line; i++) en
                break;}                                 /*   end if  *
        else {                                           /* partial re
                movetext (col+1, row+1, col+wid-2, row+h
                                        col+1, row+1+line);
                drawClear (col+1, row+1, wid-2, line, c1
                more = writText (pbuf+0, &end, col+2, ro
                                                2+line, c1, c2

                str = line = 0;
                for (i=0, end=0; i<=(hit-3); i++) end+=g
                break;}                                 /*   end else
case HOME :
        if (line == 0){
                bleep();
                break;}                                  /*   end if
        if (line> = hit-2){                              /* full rewri
                str = end = line = 0;                     /* reset tr
                drawClear (col+1, row+1, wid-2, hit-2, c
                more = writText (pbuf+str, &end, col+2,
                                                hit, c1, c2);
                break;}                                   /*   end if
        else {                                            /* partial
                movetext (col+1, row+1, col+wid-2, row+h
                                        col+1, row+1+line);
                drawClear (col+1, row+1, wid-2, line, c1
                more = writText (pbuf+0, &end, col+2, ro
                                                2+line, c1, c2

                str = line = 0;
                for (i=0, end=0; i<=(hit-3); i++) end+=g
                break;}                                 /*   end else

case PGDN :
        if (more == FALSE){
                bleep();
                break;}                                 /*   end if  */
        if ( line <= (maxlines-(hit-2))){                /* full
                line += hit-3;
                for (i=0, str=0; i<line; i++) str+=getLe
                movetext (col+1, row+hit-2, col+wid-2, r
                                        col+1, row+1);
                drawClear (col+1, row+2, wid-2, hit-3, c
                more = writText (pbuf+end, &end, col+2,
                                                hit-1, c1, c2)
                break;}                                 /*   end if  */
        else {                                          /* partial rewrit
                int nrow = ((row+hit-1)-(maxlines-line))
                tmpend=end;
                movetext (col+1, row+1+(maxlines-line),
                                        row+hit-2, col+1, row+1)
                drawClear (col+1, row+(hit-(maxlines-lin
                                        wid-2, maxlines-line, c
                while (end < size){
                        str += getLen (pbuf+str, adjwid)
                        end += getLen (pbuf+end, adjwid)
                        line ++;}                       /*   end whil
                more = writText (pbuf+tmpend, &tmpend, c
                                                wid, 2+line, c
                break;}                                 /*   end else
```

```
                case DNARROW :
                        if (more == TRUE){
                                line++;
                                for (i=0, str=0; i<line; i++) str+=getLe

                                movetext (col+1, row+2, col+wid-2, row+h
                                                row+1);
                                drawClear (col+1, row+hit-2, wid-2, 1, c
                                more = writText (pbuf+end, &end, col+2,
                                                wid, 3, c1, c2
                                break;}                 /*   end if  */
                        else {
                                bleep();
                                break;}                 /*   end else  */
                case UPARROW :
                        if (line > 0){
                                line--;
                                movetext (col+1, row+1, col+wid-2, row+h
                                                col+1, row+2);
                                drawClear (col+1, row+1, wid-2, 1, c1, 0
                                for (i=0, str=0; i<line; i++) str += get

                                more = writText (pbuf+str, &end, col+2,
                                                wid, 3, c1, c2
                                for (i=0, end=0; i<=(hit-3)+line; i++) e

                                break;}                 /*   end if  */
                        else {
                                bleep();
                                break;}                 /*   end else  */
                                }
                }
}
return;
}
```

APPENDIX 2

U S E R M A N U A L :

GRI LNG Dike Vaporization Program (DIKE)

(Version 1)

March 31, 1989

Robert F. Benenati Inc. Manhasset, N.Y.

Prologue

The computer program DIKE will run on any IBM PC/XT/AT or true compatible, equipped with the 8087 math co-processor chip and with an 80 column printer connected to the parallel port LPT-1.

Before attempting to run the program, it is recommended that you make a back-up copy of the disk and that you store the original in a safe place. Next you should examine the file CONFIG.SYS in the root directory. This file must contain the ANSI.SYS driver. If it does not, check that the file ANSI.SYS is in the root directory and use your favorite editor program to add the line:

device = ANSI.SYS

to the CONFIG.SYS file. Be sure you store the modified CONFIG.SYS file in the root directory then reboot the machine. You are now ready to run the DIKE program, which you do by just typing the word DIKE (upper or lower case) followed by the <ENTER> key.

INTRODUCTION

This personal computer-based program is designed to provide the user with vaporization rate data for LNG spills within a user-defined LNG storage dike or other impoundment. By calculating solid conductive heat transfer up thru up to three layers of dike floor and wall materials, the program can be used for evaluating the effectiveness of dike insulating alternatives in mitigating rapid vaporization of spilled LNG. Vaporization rates and volumes are provided to assist the user in determining hazard zones associated with downwind dispersion of the resulting LNG vapor cloud. Ideally, the user would use calculated vaporization rates as input to an appropriate heavy gas vapor dispersion model or laboratory experiment.

In addition, the program provides the user with the option of calculating vapor dispersion distances directly from the program, which includes a simple Gaussian passive dispersion procedure. However, this dispersion calculation should be used for comparative purposes only since, as typical of Gaussian dispersion models, it neglects important LNG vapor properties. Dispersion calculations produced by the program should not be used for hazard evaluation or for regulatory compliance evaluation purposes.

The objective of this User Manual is to provide program users with information on program organization and operation as well as underlying calculation approaches employed. User questions regarding specific aspects of the program can be addressed to either Brookhaven National Laboratory or to GRI.

PROGRAM ORGANIZATION

Two basic types of screens are employed, the selection screen and the data input screen. The selection screen will display a short list of items from which the user may choose. One of the items will be highlighted. There may be notes displayed to elaborate on the highlighted item. The arrow keys (up/down or right/left) can be used to move the highlight bar from one item to another. The RETURN key (ENTER key on some keyboards) will cause selection of the highlighted item. Once an item has been selected by pressing the RETURN key, the program will continue with the next screen. There is no provision for backing up to previous screens; use the RETURN key cautiously.

The data entry screens are for the the entry of numeric data. Each screen is self explanatory and will show all of the data required and the units expected. Generally all non-numeric keys will be deactivated while such a screen is active. The cursor will be positioned at the first data item. As numeric keys are pressed, the number represented by the key will appear on the screen. The back arrow key can be used to erase characters entered in error. Pressing the RETURN key causes acceptance of the date and moves the cursor to the next item on the screen. Once again there is no provision for backing up to previously accepted items so treat the RETURN key with respect. When the last item on a screen has been accepted, the program moves ahead to the next screen.

Whenever sufficient data has been entered, computation proceeds. That is to say computations are going on between presentation of each data screen, and in some instances, between the entry of different data items on a single screen.

On some data entry screens, with the request for specific data, reasonable default values are displayed, along with a question as to the suitability of the values presented. The question will be accompanied by a selection box with the 'yes' response highlighted. If these values are acceptable, the user may simply press RETURN and go on to the next screen. Alternatively, the user may move the highlight bar to the 'no' response before pressing RETURN and the displayed data values will disappear, the cursor will move to the first data field, and the user must then enter data values of his choosing in the usual way.

Throughout the data input process, a column of model properties is maintained on the right of the display screen. At the outset, this column of model properties is empty, but as data is accepted from the data input screens by the program, this data is inserted in the moled properties table. Thus the user has a continuous reminder of all the data already entered and accepted. This display of model properties is for information purposes only.

Occasionally, the program will detect a fatal error in the input data (such as a dike diameter smaller than the tank diameter) in time for the error to be corrected, in which case the user will be offered an oppertunity to correct the error or to abort the program. In such cases, a selection screen may overlay a data entry screen. Whenever both screen types are displayed simultaneously, the selection screen takes precedence. Incidentally, the user can abort the program at any time during the model specification phase by pressing alt-q, ie, pressing and holding the alt key then pressing the 'q' (for quit) key.

DETAIL SCREENS

Data input or problem defination is accomplished via a sequence of screens designed to simplify to the maximum extent possible the physical process of entering the data required to initialize a calculation. This chapter describes each of these input screens and provides specific instructions for their use.

The opening screen presents a very brief description of the purpose for the programm and displays a table entitled Problem Description, which is reproduced as figure 1 at the right. This table indicates all of the arbitrary user input data required to adentify a particular problem. Also shoown are the units employed for each data item. All of the data items start out blank and are filled in later as the user progresses thru the subsequent screens. This table remains on the display throughout the data input process to continuously remind the user of the specific data values which have been entered. This can be particularly usefull if part way thru the data entry process some incompatability is detected and you are offered an oppertunity to reenter the latest data item.

PROBLEM DESCRIPTION

TANK:	
diameter.....	ft
height.....	ft
Liquid height.	ft
pressure.	psi
DIKE: shape....rectangle	
style...	
construction.	
height.....	ft
dia/length...	ft
width.....	ft
wall angle...	deg
shelf.....	inch
floor.....	inch
wall.....	inch
IPC-floor....	inch
IPC-wall.....	inch
SPILL rate...	gpm
time....	min
WEATHER.....	degF
Wind speed....	m/hr
Downwind dist.	ft

figure 1

Screen 2 shown in figure 2 below refers to the dike shape and is a simple selection menu. The uppermost box is a title box indicating the overall screen purpose. The lowermost box, shown accented, indicates the user action required. The middle box indicates the selections available to the user. One of the selections is highlighted. The user can change the highlighted item by pressing the up or down arrow keys at the right of the keyboard. When the RETURN key (or ENTER key) is pressed, the highlighted item is selected and two things happen, 1) the selected item appears in the program description table on the display and 2) the screen changes to the next screen in the sequence. There is no provision for backing up to the previous screen in the sequence so treat the enter key with some respect.

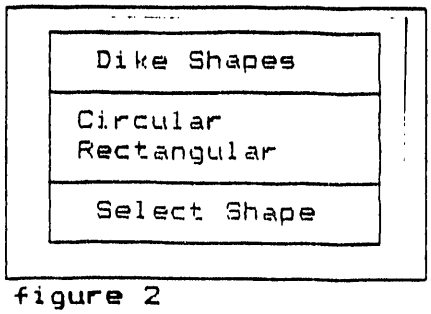


figure 2

For this screen and the two to follow, the highlight bar at the base of the screen will read "selection keys only", indicating that all other keys are temporarily inactive.

The next screen, shown in figure 3, refers to the style of construction of the dike. Three choices are available namely a dike with vertical walls, a dike with sloping walls, and a dike with sloping walls and a shelf at the junction between the dike floor and the dike walls. The floor of all dikes is assumed to be flat and level without either a drainage sump or an access ramp.

Dike Styles
Straight Sides Sloped Sides Sloped Sides w/shelf
Select Style

figure 3

Having decided on the general shape of the dike, the next screen, shown in figure 4, deals with the manner of construction of the dike. The simplest construction being just packed earth, and the more complicated being packed earth with some form of surface coating (not including the insulation). Surface coatings of crushed stone, gunite, or poured concrete are allowed. It is also possible to select a case with poured concrete on the dike floor and gunite on the dike walls, or to select a case both concrete and preexisting insulation on the dike floor and walls. If in the previous screen, a dike with straight walls was selected, a plain tamped earth dike will not be allowed nor will one with a loose rock liner be allowed since in each of these cases, the wall angle exceeds the angle of repose of the top layer of material.

Dike Construction
Tamped Earth + Gunite Earth + Poured Concrete Earth,concrete floor,gunite wall Tamped Earth Tamped Earth + Loose Rock
Select Construction Type

figure 4

All qualitative characteristics of the dike having been settled, the next screen deals with the tank and is the first screen that requires input by the user of specific numeric data. This screen is shown in figure 5 at the right. The highlight bar at the base of the screen indicates "numerics only" and infact, pressing any key other than a numeric key, the RETURN key, or the backspace key will be rejected, a short beep will be heard, and you will have a second chance to enter the data item.

Tank and Liquid Details

Diameter, ft = _

Height, ft =

Height of liquid in tank, ft =

LNG vapor pressure, psig =

Please input the values requested, following each with <RETURN>

Gas Research Institute numerics only

figure 5

For this screen, the cursor is positioned at the beginning of the first data field. After keying in the value of the tank diameter and pressing RETURN, the value entered appears in the problem description table and the cursor moves to the start of the next data field. If you key in 0.0 (see below for how to enter

such a value) for the tank diameter to simulate a tankless dike, the remainder of the screen will be skipped automatically.

Prior to pressing the RETURN key, the backspace key could have been used to erase and correct values that had been keyed improperly. Once the RETURN key has been pressed and the cursor has moved to the next data field, the numeric value keyed in has been accepted; and there is no way to change that value short of restarting the entire program. Pressing the RETURN key without first having keyed numeric values into the data field is always an error. The program traps this error and provides an opportunity to re-enter a numeric value or to abort the program. It is also possible to deliberately abort the program at any data entry point by typing ALT q (i.e., holding down the ALT key and typing q).

There are very few circumstances when a data value of 0.0 constitutes an acceptable data entry, hence such an entry results in a warning and an opportunity to re-enter a new value. If however 0.0 is really what you wanted to enter, the program will accept it and move on. To key in a value of 0.0, press the decimal point first which will result in 0. appearing in the data field. Follow this with the trailing zero and the RETURN key.

The next screen refers to specific dike dimensions and takes different forms depending on the shape and style of dike selected on earlier screens. For example with rectangular dikes, the length and width will be requested, as shown on figure 6 to the right. For circular dikes, the diameter will be requested. If a dike with sloped walls had been indicated previously, the angle of the wall from the vertical will be requested at this time. The example shown in figure 6 is for a rectangular dike with sloped sides.

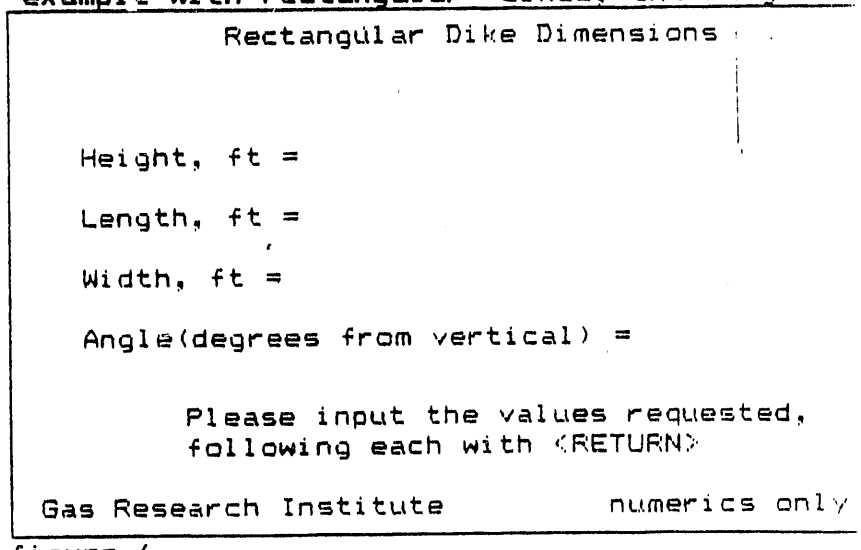


figure 6

The next screen asks for dike liner dimensions as well as new insulation dimensions. The request for liner information will only appear if a lined dike had been specified. The request for new insulation thickness will always appear for it is assumed that the user is interested in comparing the effects of boiloff both with and without insulation. Figure 7 at the top of the next page shows the screen for a dike which has been designated as having a concrete liner.

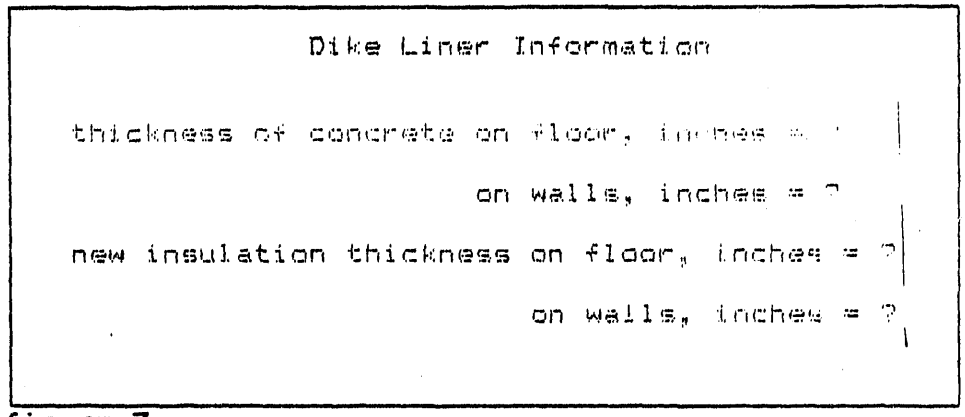


figure 7

There follows a series of screens, one for each of the materials of which the dike is constructed, showing values for density, heat capacity, and thermal conductivity. The 'tamped earth' screen is shown in figure 8 at the right. You will notice that this screen shows numerical values for each of the physical properties indicated, but asks for the users approval of these values. At this point the screen is a selection type screen since the user can only select between the choices 'yes' or 'no'. If the values displayed for these properties are acceptable, you would select 'yes' from the small selection box shown on the screen. Alternately you may select 'no' in which case, the displayed property values would disappear, the cursor will appear at the beginning of the first data field, and from this point on, the screen functions in the normal manner of a data input screen. The final screen in this series is for the insulation material used or intended to be used in the dike.

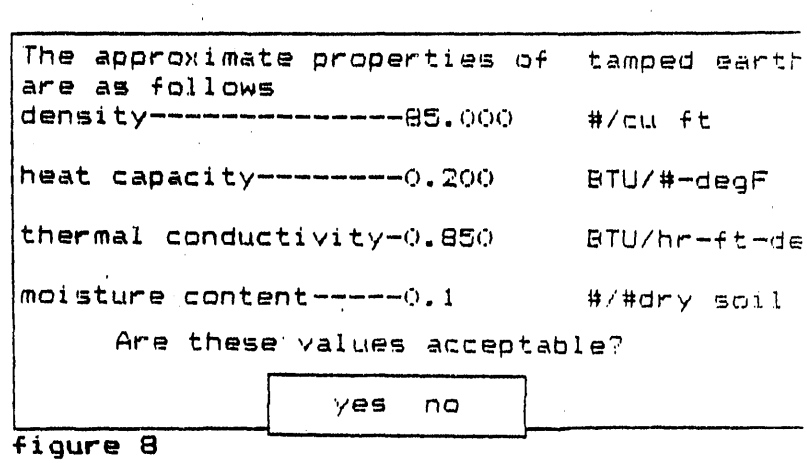


figure 8

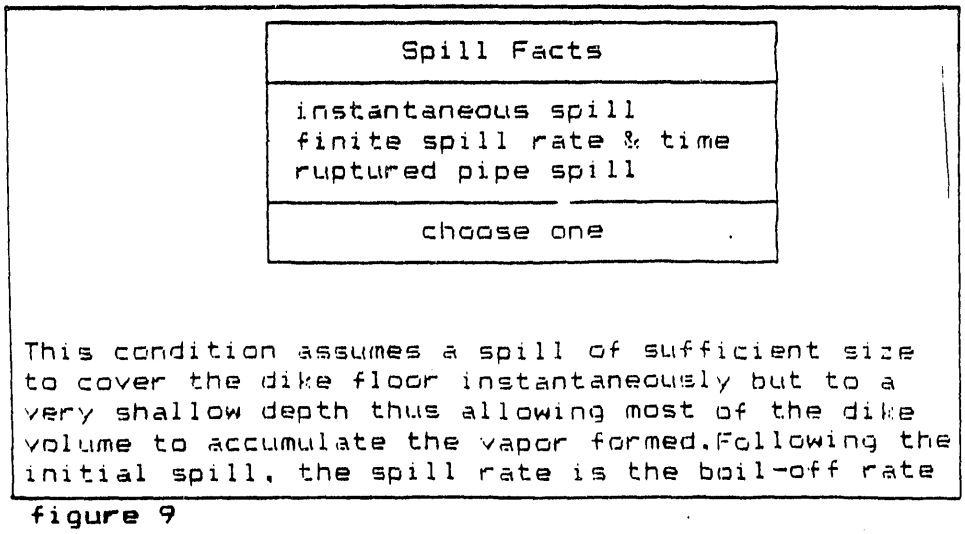


figure 9

Having described the dike in quantitative detail, we turn our attention to the nature of the LNG spill. The next screen, shown at the left, refers to the character of the LNG spill being simulated. Three selection

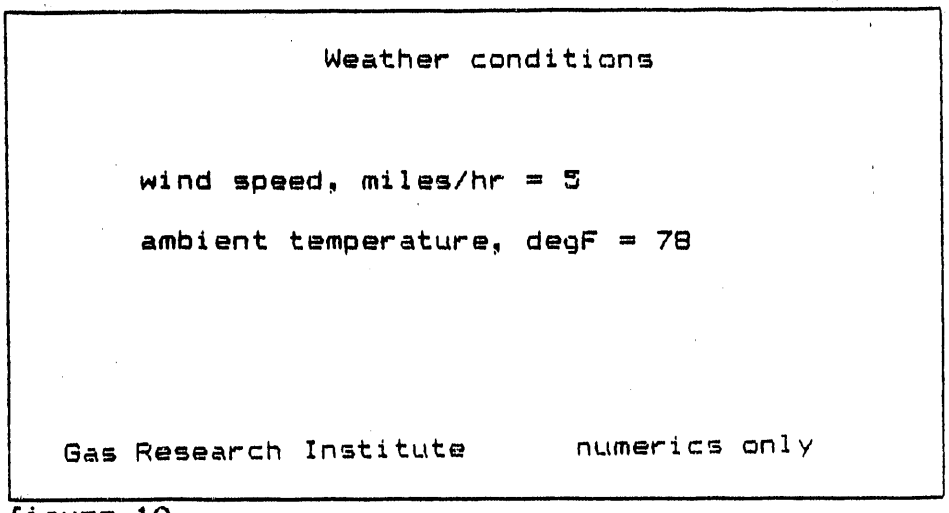


figure 10

choices are available. Highlighting each choice results in a brief description of the spill being displayed to aid in understanding the choice you are about to make. The most severe case is the instantaneous spill which assumes that the entire contents of the tank flows into the dike within a second, and a portion of it flashes instantly into vapor. Frequently this results in an instantaneous flow of vapor over the dike wall followed by the subsequent downwind dispersion.

The remaining two selection categories involve spills of a specific rate which last for a specific time. The spill is assumed to spread outward from the tank in every direction, with flashing and then evaporation occurring while the liquid is spreading. The rate of flow of the spill is user specified.

In the last selection offered, the leak is assumed to be from a rulptured pipe at the base of the tank. The pipe diameter is user specified and the leakage rate from the pipe is calculated and depends on the height of the LNG in the tank. The leakage rate is most rapid at the start and slows down as the tank empties. The duration of the spill is either for the time indicated by the user or the time necessary to empty the tank, whichever occurs first.

If calculation of the boiloff rates for both the dike, as built and for the insulated dike are completed, if vapor overflow occurred within the first fifteen minutes from the start of the leak, the user is asked if downwind disdpersion calculations are desired. A yes response results in yet another selection screen to appear. This screen is shown as figure 11 on the following page. It shows the weather conditions used by Gifford to identify six different categories which roughly indicate the degree of turbulence in the air. The user must select one of these weather categories after which the program will indicate the minimum distance at which safe concentrations of methane will be found.

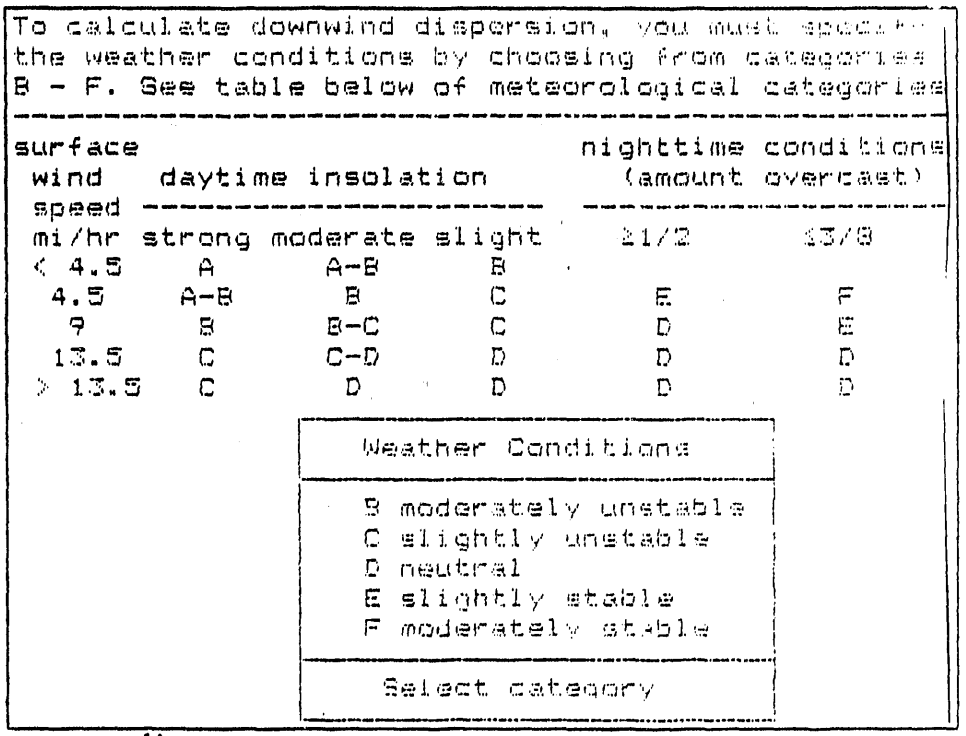

To calculate downwind dispersion, you must specify the weather conditions by choosing from categories B - F. See table below of meteorological categories
surface wind speed mi/hr
daytime insolation
strong moderate slight
nighttime conditions (amount overcast)
≥1/2 ≤3/8
< 4.5 A A-B B
4.5 A-B B C E F
9 B B-C C D E
13.5 C C-D D D D
> 13.5 C D D D D
Weather Conditions
B moderately unstable
C slightly unstable
D neutral
E slightly stable
F moderately stable
Select category

figure 11

TECHNICAL DISCUSSION

The computer program, DIKE, attempts to simulate the consequences of a spill of LNG into an impounding dike. The LNG is assumed to be in a suitable tank at a modest gage pressure (the exact pressure being supolied by the user). When a spill occurrs, some fraction of the spilled liquid is flashed to vapor due to the excess enthalpy possesed by the liquid under pressure, relative to saturated liquid at atmospheric pressure. The cold liquid spills onto the dike floor which is assumed to be at ambient temperature, i.e., more than several hundred degrees Fahrenheit above the boiling point of LNG at atmospheric pressure. The LNG recieves heat from the dike floor and boils, cooling the dike floor in the process. This cooling occurrs quite rapidly at the surface of the floor material and, as it does, the rate of heat flow into the LNG and consequently the rate of boil off of the LNG falls. Ultimately the rate of heat flow into the LNG is limited by the rate at which heat can be conducted thru the dike materials from regions below.

CONDUCTION MODEL

Heat transfer to the boiling LNG has been modeled as a one-dimensional conduction problem with a constant surface temperature boundary condition at the boiling surface and a zero temperature gradient (i.e., infinite medium) at the opposite boundary. Possible occurance of the Liedenfrost phenomena (blanketing of the surface by an insulating vapor film) has been ignored, and the surface of the dike floor is assumed to come instantly to the boiling point of the LNG, i.e., equivalent to an infinite film coefficient at the surface.

The dike construction is presumed to consist of from one to three material zones, thus up to three different material zones are permitted at the cold end of the model. The underlying material is always assumed to be tamped earth. This may or may not be covered by a liner of concrete or other suitable material, which in turn may or may not be covered by a layer of insulation. Alternately, the insulation may be applied directly to the tamped earth. The thickness of the insulation zone and of the liner zone are independantly specified by the user. The thickness of the tamped earth zone is treared as infinite. It is assumed that there exists no contact resistance between zones at their contact planes, thus at the interface between zones the temperature on each side of the interface is the same and the heat flux across the interface is identical on each side of the interface.

All dike structures modeled are assumed to be at ambient temperatur at the start of the spill. The temperature of the surface in contact with the spill is assumed to fall instantly to the LNG boiling point. The temperature immediately under the surface and indeed the temperatures throughout the dike construction materials follow the laws of conduction heat transfer.

There are three specific cases of interest in the present situation, namely:

i) An unlined, uninsulated dike, e.g., a dike composed of a single homogeneous material.

ii) An insulated, unlined dike or a lined, uninsulated dike, e.g., a dike composed of two layers of different materials.

iii) An insulated, lined dike, e.g., a dike composed of three layers of different materials.

i) Single Material Semi-infinite Thickness Case

The dike is considered to be a semi-infinite solid bounded by the xy plane only and extending to infinity in the positive x direction. The initial temperature, TO, is assumed to be uniform in the solid and to be at the ambient temperature. At time = 0, the temperature of its surface at x = 0 is suddenly changed to and maintained at Ts, the boiling temperature of LNG.

This is a classical conduction problem which is described extensively in the literature. The temperature at a depth x into the structure at any time following the spill is given by [6]

$$(T - Ts) / (TO - Ts) = \text{erfc}\ (X)$$

where erfc stands for the error function or probability integral and $X = x / 2\sqrt{\alpha\theta}$. The instantaneous heat flux at the surface is then given by

$$q / A = K\ (TO - Ts) / \sqrt{\pi\alpha\theta}$$

and the surface cumulative heat flux is given by

$$Q / A = 2k\ (TO - Ts)\ \sqrt{\theta/\pi\alpha}$$

ii) The Composit Solid or Lined Dike Case

The dike is considered to be a semi-infinite solid as before but now the distance x = 0 and x = 1 is assumed to be a material different from the bulk of the solid, as for example a layer of concrete over tamped earth. As before, the temperature isd everywhere assumed to be the ambient temperature and at time = 0, the surface temperature at x = 0 is suddenly changed to the temperature of the boiling LNG.

It is assumed that there is no contact resistance between the two materials at x = 1. This results in the requirement that at x = 1, $T_1 = T_2$ (subscripts 1 and 2 refer to the two different materials), and $k_1 \delta T_1 / \delta x_1 = k_2 \delta T_2 / \delta x_2$ for all time > 0.

This case too has been studied extensively and is described in the literature [2]. The local temperature in each of the two materials is given by:

$$\frac{T_1 - T_s}{T_0 - T_s} = \sum_{n=0}^{\infty} \beta^n \left\{ \operatorname{erfc} \frac{(2n+1)\ell + x}{2\sqrt{\alpha_1 \theta}} - \beta \operatorname{erfc} \frac{(2n+1)\ell - x}{2\sqrt{\alpha_1 \theta}} \right\}$$

$$\frac{T_2 - T_s}{T_0 - T_s} = \frac{2}{1+\sigma} \sum_{n=0}^{\infty} \beta^n \operatorname{erfc} \frac{(2n+1)\ell + \gamma x}{2\sqrt{\alpha_1 \theta}}$$

where $\gamma = \sqrt{\alpha_1/\alpha_2}$, $\sigma = \frac{k_2 \gamma}{k_1}$, $\beta = \frac{\sigma - 1}{\sigma + 1}$

The temperature gradient at the surface is given by:

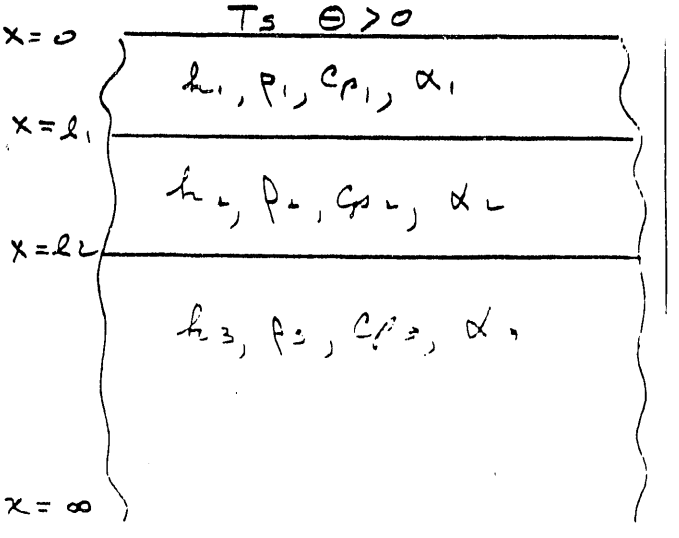
$$\left.\frac{\partial T_1}{\partial x}\right|_{x=0} = \frac{(T_0 - T_s)}{\sqrt{\pi \alpha_1 \theta}} \left\{ 1 + 2 \sum_{n=1}^{\infty} \beta^n e^{-\frac{n^2 \ell^2}{\alpha_1 \theta}} \right\}$$

and the cumulative heat flux is given by [2]

$$Q/A = \frac{2 k_1 (T_0 - T_s)}{\sqrt{\pi \alpha_1}} \left\{ 1 + 2 \sum_{n=1}^{\infty} e^{-\frac{n^2 \ell^2}{\alpha_1 \theta}} - \frac{n\ell}{\sqrt{\alpha_1 \theta}} \operatorname{erfc} \frac{n\ell}{\sqrt{\alpha_1 \theta}} \right\}$$

iii) Multi-Layered Solid or Insulated Lined Dike Case

Here too the dike is considered to be a semi-infinite solid but now there are two layers of finite thickness, each composed of a different material, on top of the semi-infinite earth zone as shown schematically below:



This case is not treated in the literature on conduction heat transfer, but it can be handled readily using the classical

techniques of numerical analysis wherein the model geometry is discretized and conduction equations and energy balance equations are solved at each node resulating from the discretization.

While this problem solving technique is well known, it posseses a well known shortcoming which makes it less desirable than the closed-form solutions presented above. To accurately follow the rapid temperature changes occurring close to the surface, very small nodal distances must be chosen in that region and the small distances mandate a very small time step else the calculations become wildly unstable. Solutions in this mode become quite time consuming to the point of trying ones patience.

In the specific case where this mode of calculation is called for, the first two nodes closest to the boiling liquid have been taken to be only 0.01 inch thick. The next two nodes have been taken to be 0.02 inches thick. In this way a total of eight nodes have been crammed into the first 0.1 inch. Node dimensions increase as the distance from the boiling liquid increases but it is the two closest to the surface that mandate the time step used in the calculations and which cause this calculation to proceed very slowly.

The program DIKE uses whichever of the above described procedures is appropriate at each phase of its calculations.

DOWNWIND DISPERSION

The portion of this program which deals with the dispersion of the vapor cloud resulting from the LNG spill calculates the farthest downwind distance at which a methane concentration at or above 2.5% will be found. The calculation is based on the continuous line source model and uses the maximum rate of evaporation found in the dike heat transfer section together with a user specified set of atmospheric conditions.

The degree of dispersion of the vapor cloud as it moves downwind depends on the stability of the atmosphers, i.e., the degree of turbulence or gustiness in the atmosphere. Dispersion is maximized, and therefore the methane concentration reduced, when the atmosphere is unstable. The most popular atmospheric dispersion model is derived by considering statistical variations around the mean concentration value and results in the so called Gaussian distribution model which gives the downwind concentration as:

$$C = Q_L * Z^* * Y^* / V$$

$$Z^* = 1/\sqrt{\pi}\sigma Z \; e^{-(z-h)/(\sqrt{2}\sigma z)2} + e^{-(z+h)/(\sqrt{2}\sigma z)2}$$

$$Y^* = \tfrac{1}{2}(\mathrm{erf}(\mathrm{dikewidth} - y)/2\sqrt{2}\sigma y + \mathrm{erf}(\mathrm{dikewidth}+y)/2\sqrt{2}\sigma y)$$

In this program, the crosswind distance, y, is taken as zero thus maximizing the methane concentration at anyplace downwind.

The dispersion parameters, σ_z and σ_y, depend on both the atmospheric conditions and the downwind distance. They are herein calculated from:

$$\sigma_y = con1 * 3.2808 * (xlee/3280.0)^{con2}$$

$$\sigma_z = 3.2808 + 10^{con3 + con4 * \ln(xlee/3280) + con5 * \ln(xlee/3280)^2}$$

where the five coefficients are different for each atmospheric condition as given in the table below. The atmospheric categories listed in this table are those of Gifford[4] and the specific constants as well as the form of the correlations have been taken from the report entitled "LNG Safety Program Phase II Consequences of LNG Spills on Land[1]".

Table of Dispersion Constants

Weather Category	con1	con2	con3	con4	con5
B	158.0	0.9	2.041	1.048	0.041
C	104.0	0.913	1.786	0.914	0.0
D	69.0	0.919	1.505	0.737	-.105
E	51.0	0.919	1.332	0.678	-.112
F	34.0	0.919	1.146	0.65	-.113

BIBLIOGRAPHY

1. BMI Report, LNG Safety Program Phase II Consequences of LNG Spills on Land, Nov 1973, Appendix D

2. Carslaw, H.S. & Jaegar, J.C., Conduction of Heat in Solids, Oxford University press, 1947 p321-322

3. Fontana, J. et al, Development of an Insulating Polymer Concrete Overlay for Dike Insulation at Long Island Lighting Company's LNG Storage Facility, BNL Report Dec 1987

4. Gifford, F.A., Use of Routine Meteorogical Observations for Estimating Atmospheric Dispersion. Nuclear Safety 2, 47 (1961)

5. Neville, A.M., Hardened Concrete Physical & Mechanical Aspects, Iowa State University Press, Ames Iowa. 1971

6. Schneider, P.J., Conduction Heat Transfer, Addison-Wesley Publishing CO., 55 p240-242

END

DATE
FILMED
3/9/92