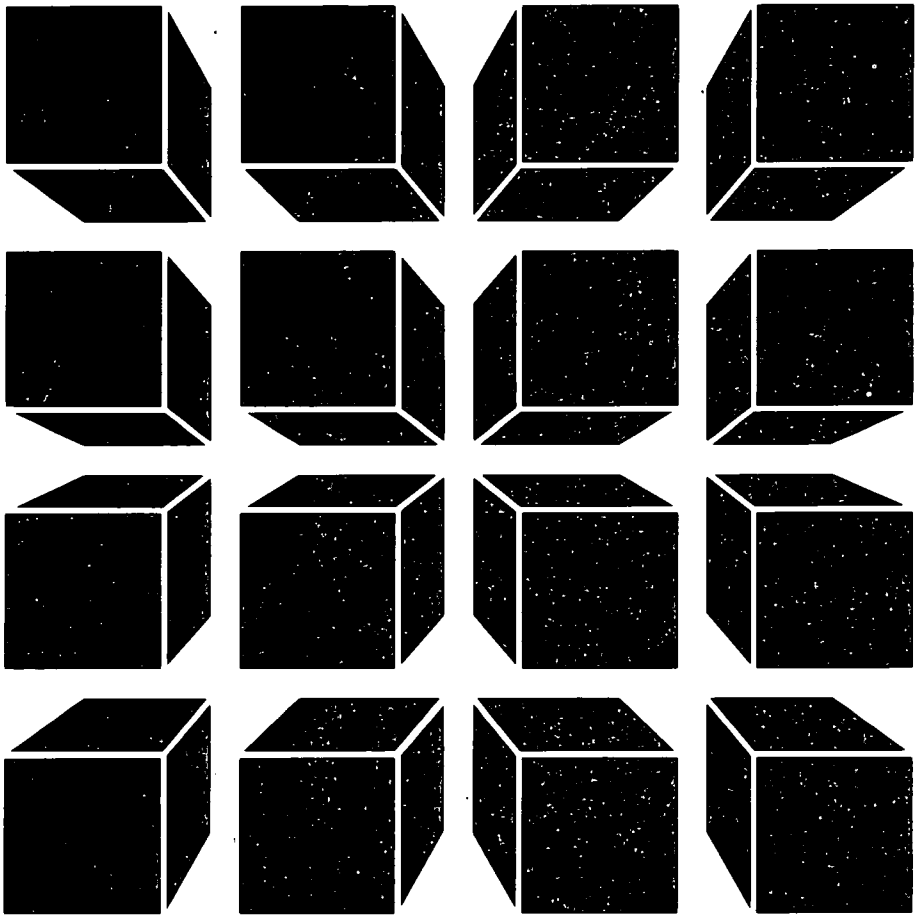
AT&T VIDEOTAPE LIBRARY

C LANGUAGE FOR PROGRAMMERS



WORKBOOK VOL II

AT&T

WORKBOOK VOLUME II

CONTENTS

VOLUME 8—Introduction to Pointers

Declaring, Initializing, and Using Pointers 8-2
Volume 8 Exercise - Pointers & and * 8-5
Volume 8 Exercise - Answers 8-7
Library Functions That Return Pointers 8-8
Converting an ASCII String to Integer, Long or Double 8-10
Volume 8 Summary 8-12
Volume 8 Lab 8-14
Volume 8 Lab - Solutions 8-18

VOLUME 9—Structures and Unions

Establishing a Template and Declaring a Structure 9-2
Referencing Structure Members - Dot Operator 9-4
Volume 9 Exercise - Structures 9-5
Volume 9 Exercise - Answers 9-7
Pointers to Structures, the -> Operator 9-8
Initializing External and Static Structures 9-10
Unions vs. Structures 9-12
Volume 9 Summary 9-14
Volume 9 Lab 9-16
Volume 9 Lab - Solutions 9-21

VOLUME 8

Introduction to Pointers

DECLARING, INITIALIZING AND USING POINTERS

```
#include    <stdio.h>
main()
{
   int  num1=3, num2=6;
   int  *p;

   p = &num1;
   printf("%d\n",*p);

   *p = 20;
   printf("%d ",*p);
   printf("%d\n",num1);

   p = &num2;
   printf("%d\n",*p);
}
```

Output:

```
3
20 20
6
```

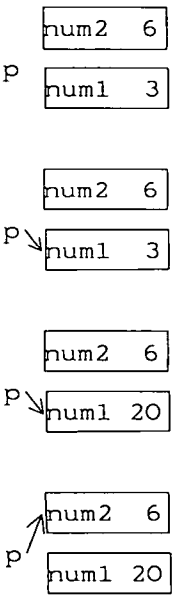
Hypothetical Stack

Addr.	*Variable*	*Value*
502	p	?
501	num2	6
500	num1	3

Addr.	*Variable*	*Value*
502	p	500
501	num2	6
500	num1	3

Addr.	*Variable*	*Value*
502	p	500
501	num2	6
500	num1	20

Addr.	*Variable*	*Value*
502	p	501
501	num2	6
500	num1	20



VOLUME 8 EXERCISE
POINTER & and *

3. What is wrong with this program?

```
1       #include    <stdio.h>
2
3       main()
4       {
5         int    num1 = 100, *p;
6
7         printf("num1 is %d\n", num1);
8         printf("*p is %d\n", *p);
9       }
```

Will it compile?
Will it run?
What will be printed?

4. Suppose that **i** is an integer and **p** is an integer pointer. What does the following phrase mean?

p "points to" i

VOLUME 8 EXERCISE - ANSWERS
POINTER & and *

1. count = 10, x = 10
2. i1 = 5, i2 = 12, *p = 5, *q = 5
3. The program will compile and run. The output is indeterminate since the pointer p MUST be initialized.
4. p's value is the address of i.

LIBRARY FUNCTIONS THAT RETURN POINTERS

Some String-Handling Functions

SYNOPSIS

```
#include   <string.h>

char   *strcat(s1, s2)
char   *s1, *s2;

int    strcmp(s1, s2)
char   *s1, *s2;

char   *strcpy(s1, s2)
char   *s1, *s2;

char   *strncpy(s1, s2, n)
char   *s1, *s2;
int    n;

int    strlen(s)
char   *s;
```

DESCRIPTION

strcat appends s2 to s1 and returns s1.

strcmp's return value is less than, equal to, or greater than 0, if s1 is lexicographically less than, equal to, or greater than s2.

strcpy copies s2 to s1, stopping after the null is copied; s1 is returned.

strncpy copies exactly n characters from s2 to s1, truncating s2, or adding null characters to s1 if necessary.

strlen returns the number of non-NULL bytes in s.

CONVERTING AN ASCII TO INTEGER, LONG OR DOUBLE

Check the local documentation for function behavior when an underflow or overflow is detected. The code shown below is one way of handling errors when using atof( ) on a UNIX System. The return value of scanf() need not be checked when %s is used because it will never fail, except for end-of-file.

```
#include      <errno.h>
#include      <ctype.h>
#include      <math.h>
main()
{
   int    x;
   char   line[257];
   double atof(),d;
   while (1) {
        errno = 0;
        printf("Enter a real number: ");
        scanf("%256s",line);
        d = atof(line);
        if ( (d == HUGE || d == 0) && (errno == ERANGE) )
                printf("\tError: Under/Overflow\n");
        else if (!(isdigit(line[0])))
                printf("\tError: Non-digit \n");
        else    break;
        }
    printf("Number entered: %g \n",d);
}
```

There are advantages to using these functions instead of scanf(). scanf()'s many conversion characters are hard to remember (%d, %hd, %ld, %f or %lf). Programmers often forget to pass an address to scanf(), and the input stream must be cleared when bad input is detected.

VOLUME 8 SUMMARY

Null pointers

A null pointer has a 0 value.
It is illegal to read from or write to a null pointer.
NULL is defined in stdio.h

Some functions that use pointers:

Function	Notes	UNIX Manual Page
gets()	#include <stdio.h>	gets(3S)
strcat() strcmp() strcpy() strlen()	#include <string.h>	string(3C)

Accessing Command Line Arguments

```
main(argc, argv)          $ a.out  file1  file2
int  argc;                argv[0] argv[1] argv[2]
char *argv[];
{
...
}
```

3. Accessing array contents.

   a. Write a function that returns the sum of the elements of an array of integers. It should accept two arguments: the name of the array and the number of elements. Use the * operator when summing the array elements. If you find this difficult, write the function first using the [ ] operator, then rewrite it using the * operator.

      You can test the function this way:

```
main()
{
        static int table[ ] = { 3, 7, 10, 6, 4 };

        printf("Sum is %d\n", sum(table,5));
}
```

   b. Write a similar function which returns the sum of the elements of an array of doubles.

6. Same as problem 5, but if two arguments after the program name are supplied, print an error message if they are identical. (This is similar to what the UNIX cp command does before it copies a file).

Building Block Lab

7. In Volume 4 [Lab Problem 4.3(b)], you wrote a program that would prompt the user for names and telephone numbers, store the data in an array, and print the stored data on the terminal.

   You have given it to some office personnel to test. Some users reported flaws in the program. Specifically, it does not properly handle the entry of a last name like *De Luca* because *scanf* sees the blank character as termination of the *%s* conversion requested. Also, users have requested provision for street address, city, state, and zip code data.

   Rewrite the 4.3(b) program to address these concerns.

VOLUME 8 LAB - SOLUTIONS

```
/* 8.2.c */
/* This program tests an absolute value function. */
/* The address of an integer is passed to the function which */
/* changes the integer to its absolute value if it is negative. */

#include <stdio.h>
#define TRUE 1

main()
{
        int     num;
        void    abs();

        printf("This program will print the absolute value\n");
        printf("of the integers entered.  To quit, hit \n");
        printf("the DELETE key.\n\n");
        while (TRUE) {
                printf("Enter an integer: ");
                if (scanf("%d", &num) != 1) {
                        printf("Illegal input. Try again.\n\n");
                        while (getchar() != '\n')
                                ;
                        continue;
                        }
                printf("The absolute value of %d is ", num);
                abs(&num);
                printf("%d\n\n",num);
                }
}

void abs(p)
int     *p;
{
        if ( *p < 0 )
                *p = -*p;
}
```

VOLUME 8 LAB - SOLUTIONS

```
/* 8.4a.c - sncat() */

/* Appends at most n character of s2 to s1.  Returns s1. */

char *
sncat(s1, s2, n)
register char *s1, *s2;
register n;
{
   register char *p;

   p  = s1;
   while(*p != '\0')   /* Find end of s1 */
        p++;
   while((*p = *s2) != '\0')  /* Copy chars till NULL reached */
        if(--n < 0) {   /* or n characters copied */
             *p = '\0';
             break;
             }
         else {
             p++;   /* p traverses the first string */
             s2++;  /* s2 traverses the second */
            }
    return(s1);
}
```

VOLUME 8 LAB - SOLUTIONS

```
/* 8.5.c */
/* Prints error message if there are not exactly 2 */
/* arguments after the program name on the */
/* command line. */

#include <stdio.h>

main(argc, argv)
int     argc;
char    *argv[];
{
   if ( argc != 3 ) /* program name + 2 more words */
          printf("%s: two arguments expected.\n",argv[0]);
}
```

```
/* 8.6.c */
/* Prints error message if there are not exactly 2 */
/* arguments after the program name on the command */
/* line. Otherwise, prints error message if the two */
/* arguments are the same. */

#include <stdio.h>

main(argc, argv)
int     argc;
char    *argv[];
{
  if ( argc != 3 ) /* program name + 2 more words */
          printf("%s: two arguments expected.\n",argv[0]);
  else if (strcmp(argv[1], argv[2]) == 0)
          printf("%s: two arguments identical.\n",argv[0]);
}
```

```
/* 8.7.c (cont.) */

        printf("\n\nPlease enter information in response to prompt.");

        while ( i < MAX_ENTRIES ) {
           printf("\nEnter Last Name(20 char. max.): ");
           gets (linebuf);
           strncpy (lastname[i], linebuf, 20);
           printf("\nEnter First Name and M.I.(18 char. max.): ");
           gets (linebuf);
           strncpy (firstname[i], linebuf, 18);
           printf("\nEnter Phone Number(15 char. max.): ");
           gets (linebuf);
           strncpy (phone[i], linebuf, 15);
           printf("\nEnter Street address(25 char. max.): ");
           gets (linebuf);
           strncpy (address[i], linebuf, 25);
           printf("\nEnter City & State(20 char. max.): ");
           gets (linebuf);
           strncpy (city_st[i], linebuf, 20);
           printf("\nZip(10 char. max.): ");
           gets (linebuf);
           strncpy (zip[i], linebuf, 10);
           printf("\nAnother entry?? (y or n): ");
           gets (linebuf);
           if ( linebuf[0] != 'y' ) break;
           i++;
           }
        for ( j = 0; j <= i; ++j ) {
           printf("%-20s %-18s %-15s\n", lastname[j], firstname[j],
                    phone[j]);
           printf("%-25s %-20s %-10s\n", address[j],city_st[j], zip[j]);
           }
     }
```

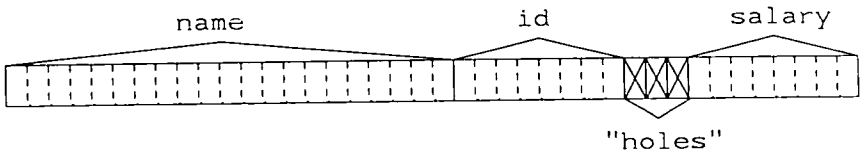
VOLUME 9

Structures and Unions

Establishing a Template and Declaring a Structure

struct *[tag]* { *member-list* };

```
1       struct emp  {
2                       char   name[21];
3                       char   id[8];
4                       double  salary;
5       };  /* template - no storage reserved */
6
7       main()
8       {
9               struct emp  prgmr;      /* reserves storage */
10              int     num;
11              ...
12      }
13
14      f()
15      {
16              struct emp supervisor; /* reserves storage */
17              ...
18      }
```



VOLUME 9 EXERCISE
STRUCTURES

```
/* Program to track tickets sales */ #include  <stdio.h>

struct t_info {
        float price;
        int   sold;
};
main()
{
        struct  t_info      ticket;
        int     num;

   (E)                                  /* Initializes price to 6 dollars */

   (F)                                  /* Initializes number sold to 0 */

   (G)                                  /* Adds 5 to number of tickets sold */

        /* Line below prints total amount collected */

   (H)  Printf("%.2f\n",              );
}
```

(A) (B) (C) (D)

1. Match labels with the correct description.

   A P - Members or fields of **struct t_info**

   B Q - Declares and reserves storage for one structure

   C R - Tag, allows subsequent declarations of this type

   D S - Structure template, describes members of the **struct t_info** type, does not reserve storage

2. Fill in the statements labeled E, F, G, and H.
3. The data type of **num** is **int**. What is **ticket**'s data type?

VOLUME 9 EXERCISE - ANSWERS
STRUCTURES

1. A—R, B—P, C—S, D—Q
2. E – ticket.price = 6.0;
   F – ticket.sold = 0;
   G – ticket.sold += 5;
   H – ticket.price * ticket.sold
3. struct t_info

Pointers to Structures, the -> operator

```
1     /* Prints total of employee salaries */
2     #include  <stdio.h>
3     #define             NUM_EMPS            100
4
5     struct emp  {
6           char   name[21];
7           char   id[8];
8           double  salary;
9     };
10
11    main()
12    {
13        struct emp staff[NUM_EMPS], *sp;
14        double          sal_tot = 0;
15
16        fillarray(staff); /* Fills array with data */
17        sp = staff;
18        for ( sp = staff; sp != staff[NUM_EMPS]; sp++)
19                        sal_tot += sp->salary;
20        printf("Total of salaries: $%.2f\n", sal_tot);
21    }
```

Initializing External and Static Structures

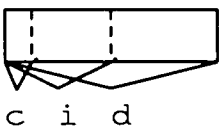
```
1     struct course {
2          char   name[30];
3          int    number;
4          char   nickname[30];
5     } title = { "C for Experienced Programmers",
6                   1001,
7                  "The Semicolon Odyssey" };
8     struct mailinfo {
9           char     name[25];
10          char     mail_addr[30];
11    } proj_member[] = {
12                          { "Jean Griffin",  "rz3bb!jmg"},
13                          { "Hamid Assous", "rz3bb!ha"},
14                          { "Ken Lewis",  "rz3ba!kal"},
15                          { "Mike McLoughlin", "rz3bb!tmm"},
16                          { "Ella Palizi",  "rz3bb!ep"},
17                          { "Frank Prihoda", "rz3bb!fjp"},
18                          { "Tom Tatem",     "rz3bb!tct"},
19                          { "",              ""},
20    };
21    f()
22     {
23       static struct mailinfo admin =
24                 {"Administrator","root"};
25       ...
26     }
```

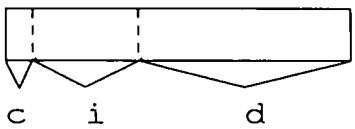
Unions vs. Structures

- Declaration, tag, template same as structures except keyword **union** used.
- Can hold only one member at a time
- Big enough to hold largest member

```
union u_tag {
        char    c;
        int     i;
        double  d;
} u_item;
```

```
struct s_tag {
        char    c;
        int     i;
        double  d;
} s_item;
```






```
printf("The size of u_item is %d\n", size of(u_item));
printf("The size of s_item is %d\n", size of(s_item));
```

Output:

```
8
16
```

Size of Operator

yields the number of bytes in a variable or data type (machine dependent).

```
int     x;
char    line[257];

printf("%d\n", sizeof(x));/* size of an int, machine dependent */
printf("%d\n", sizeof(int));   /* size of an int, machine
                                  dependent */

printf("%s\n", sizeof(line));  /* 257, regardless of contents */
```

3. Put each of the major parts of the program from Problem 3 into separate functions.
4. Change the solution in Problem 4 to use a structure pointer when filling the array and printing its contents.
5. Add a line to the output that prints the total value of the inventory.

Insurance Company Database Problem

6. The header file "insurance.h" shown on the next page contains a structure template that is to be used to access the policy holder underwriting database for an insurance company. The **client** template contains members that will hold information about a client of the insurance company. A client is classified as being either a car driver or a truck driver.

7. Hopewell S&L has kept its account records on punched cards. Management has asked you to design a computer database to replace the card files holding customer accounts. Your first task is to design a C Language structure to hold the required data for an account record. Name the structure tag *acct_rec*. So that the structure template may be shared by using programs, create it in a header file named **acct_rec.h**.

   In your discussions with S&L office staff, you have learned the following design requirements:

   Customer Information:

   There are two types of account holders: the general public and employees of Amwell Valley Technologies. An account is either a general public account or an AVT account. Mailing address data for the general public should include separate fields for:

   Street address
   City and state
   Zip code

   Address data for AVT employees should include:

   Department number
   Company site location

   To aid address label design, separate fields are required for last names and first names and initials. Also, space should be provided for a telephone number and an account number which is a 5-digit integer type number.

VOLUME 9 LAB - SOLUTIONS

```
/* 9.1.c */
/* Reads an item's name, quantity in stock, and price into */
/* a structure, then prints the structure's contents. */

#include      <stdio.h>

struct item {
        char   name[31];
        int    quantity;
        double     price;
};
main()
{
        struct item     part;
        double atof();
        int atoi();
        char    buf[257];

        printf("\nItem name: ");
        gets(part.name);
        printf("Quantity in stock: ");
        gets(buf);
        part.quantity = atoi(buf);
        printf("Price: ");
        gets(buf);
        part.price = atof(buf);
        printf("\n\n");
        printf("Item name: %s\n",part.name);
        printf("    price: $%.2f\n",part.price);
        printf(" quantity: %d\n",part.quantity);
}
```

```
/* 9.2.c */
/* Reads the name, quantity in stock, and price of */
/* NUM_ITEMS items into an array of structures, then */
/* prints the array contents. */

#include      "inventory.h"
#include      <stdio.h>

main()
{
        struct item     part[NUM_ITEMS];
        double atof();
        int     atoi();
        char    buf[257];
        int i;

        for (i = 0; i < NUM_ITEMS; i++) {
                printf("\nItem name: ");
                gets(part[i].name);
                printf("Quantity in stock: ");
                gets(buf);
                part[i].quantity = atoi(buf);
                printf("Price: ");
                gets(buf);
                part[i].price = atof(buf);
                }
        printf("\n\n\t\tINVENTORY REPORT\n\n");
        printf("    NAME          PRICE      QUANTITY\n\n");
        for (i = 0; i < NUM_ITEMS; i++) {
                printf("%15s",part[i].name);
                printf("%11.2f",part[i].price);
                printf("%13d\n",part[i].quantity);
                }
}
```

```
/* 9.3.c (cont.) */

void print_report()
{
        int     i;

        printf("\n\n\t\tINVENTORY REPORT\n\n");
        printf("    NAME          PRICE      QUANTITY\n\n");
        for (i = 0; i < NUM_ITEMS; i++) {
                printf("%15s",part[i].name);
                printf("%11.2f",part[i].price);
                printf("%13d\n",part[i].quantity);
                }
}
```

```
/* 9.4.c (cont.) */

void print_report(p, count)
struct item *p;
{
        struct item *end;

        printf("\n\n\t\tINVENTORY REPORT\n\n");
        printf("    NAME          PRICE       QUANTITY\n\n");
        for (end = p + count; p < end; p++) {
                printf("%15s",p->name);
                printf("%11.2f",p->price);
                printf("%13d\n",p->quantity);
                }
}
```

```
/* 9.5.c (cont.) */

void print_report(p, count)
struct item *p;
{
        struct item *end;

        printf("\n\n\t\tINVENTORY REPORT\n\n");
        printf("    NAME          PRICE       QUANTITY\n\n");
        for (end = p + count ; p < end; p++) {
                printf("%15s",p->name);
                printf("%11.2f",p->price);
                printf("%13d\n",p->quantity);
                }
}

double inv_value(p, count)
struct item *p;
{
        struct item *end;
        double  sum;

        for ( sum = 0, end = p + count; p < end; p++)
                sum += p->price * p->quantity;
        return(sum);
}
```

```
/* 9.7 */
/* acct_rec.h */
struct acct_rec {
        char lastname[21];
        char first_mi[19];
        char phone[16];
        short acct_num;
        char type;      /* e(AVT emp.) or p(public) */
        union {
           struct {
                char st_addr[26];
                char city_st[21];
                char zip[11];
           } mail_add;          /* Gen. public */
           struct {
                char dept[11];
                char location[21];
           } co_addr;           /* AVT employee */
        } address;
        double pb_bal; /* Passbook savings balance */
        short num_loans;        /* Number of outstanding loans */
        struct {
           char type;   /* a, m, p, s or c */
           char date[9];        /* Date of loan acct. */
           double principl;     /* Amount borrowed */
           float rate;          /* Int. rate */
           double curr_bal;
           double past_due;     /* Over due loan payment */
        } loans[5];
};
```

VOLUME 10

File Input/Output

OPENING A FILE - fopen()

SYNOPSIS
```
#include  <stdio.h>
FILE  *fopen( file-name, type )
char  *file-name, *type;
```

DESCRIPTION The file named is opened according to *type* which may be

"r"	"r+"
"w"	"w+"
"a"	"a+"

Returns NULL on failure.

EXAMPLE

```
1    #include      <stdio.h>
2
3    main()
4    {
5        FILE    *fp;
6
7        fp = fopen("logfile", "w");
8        if (fp == (FILE *)NULL)
9              printf("Open failed\n");
10       ...
11   }
```

A file is opened using fopen(). The first argument to fopen() is the address of a string containing the pathname of the file. The second argument is the address of a string specifying how the file is to be opened, i.e., "r", "w", etc. The function returns a FILE pointer which is assigned to a variable, in this case to **fp**. After a file is opened, it is accessed using the FILE pointer; the file name is no longer used.

The return code from fopen() should be checked to insure the open operation was successful.

SAMPLE PROGRAM USING
fopen() and fclose()

```
1       /* This program copies a file, */
2       /* argv[1] is copied to argv[2] */
3
4       #include      <stdio.h>
5
6       main(argc, argv)
7       int   argc;
8       char  *argv[];
9       {
10            FILE   *rp, *wp;
11
12            if (argc < 3){
13                  printf("2 FILE NAMES REQUIRED\n");
14                  exit(1);
15                  }
16            if ((rp=fopen(argv[1],"r")) == (FILE *)NULL) {
17                  printf("Can't open %s\n",argv[1]);
18                  exit(2);
19                  }
20            if ((wp=fopen(argv[2],"w")) == (FILE *)NULL) {
21                  printf("Can't open %s\n",argv[2]);
22                  fclose(rp);
23                  exit(3);
24                  }
25            /* Read/write functions shown later */
26            fclose(rp);
27            fclose(wp);
28      }
```

VOLUME 10 EXERCISE - ANSWERS
fopen()

1. It is declared in stdio.h

2a) "r"

b) "a"

b) "r+"

b) "w"

CHOOSING THE APPROPRIATE READ/WRITE FUNCTION

SYNOPSIS (Contd)

```
int     fscanf(stream, format [ ,pointer ] ...)
FILE    *stream;
char    *format;

int     fprintf(stream, format [ ,arg ] ... )
FILE    *stream;
char    *format;

int     fread(ptr, size, nitems, stream)
char    *ptr;
int     size, nitems;
FILE    *stream;

int     fwrite(ptr, size, nitems, stream)
char    *ptr;
int     size, nitems;
FILE    *stream;
```

CHARACTER I/O: ungetc()

SYNOPSIS

```
int ungetc(c, stream)
int    c;
FILE   *stream;
```

DESCRIPTION Puts one character back on input stream providing the stream was previously read. Next read operation will read that character.

EXAMPLE

```
/* Skips over any number of white space */
/* characters in input file */
void
skip_whites(fp)
FILE *fp;
{
        int    c;
        while ((c = fgetc(fp)) == ' ' ||
                c == '\t' || c == '\n')
                ; /* Null loop body */
        ungetc(c, fp);  /* Put non-white back */
}
```

RANDOM ACCESS - fseek() and ftell()

SYNOPSIS

```
int fseek(stream, offset, ptrname) FILE  *stream;

long  offset;
int  ptrname;
long  ftell(stream)
FILE  *stream;
```

DESCRIPTION

fseek sets position of next read/write operation on the *stream*, *offset* bytes from *ptrname* which has the values:

0	top
1	current
2	end

Returns non-zero for improper seeks, otherwise 0.

ftell returns offset of current byte from top of file.

EXAMPLES

```
fseek(fp, 0L, 0);/* rewind */

fseek(fp, 0L, 2);/* bottom of file */

                /* back up one structure */
fseek(fp, -(long)sizeof(struct emp), 1);
```

ERROR HANDLING EXAMPLES

The first example shown below will copy the contents on one file to another. The loop will terminate when **EOF** is reached on the input file.

The second example will attempt to read an input file several times before it gives up.

```
while(1) {
        n = fread( buf, BUFSIZE, 1, rp);
        if ( feof(rp)
                break;
        fwrite(buf, n, 1, wp);
        }
```

```
        /* Try MAX times to read device */
        success = 0;
        clearerr(rp);
        fread( buf, BUFSIZE, 1, rp);
        for (i = 1; i < MAX; i++) {
                if ( !ferror(rp) ) {
                                success = 1;
                                break;
                                }
                fread( buf, BUFSIZE, 1, rp);
                }
        if (success)
                printf("read successful\n");
        else
                printf("Max retries - read failed\n");
```

SYNOPSIS (Contd)

```
int     fscanf(stream, format [ ,pointer ] ...)
FILE    *stream;
char    *format;

int     fprintf(stream, format [ ,arg ] ... )
FILE    *stream;
char    *format;

int     fread(ptr, size, nitems, stream)
char    *ptr;
int     size, nitems;
FILE    *stream;

int     fwrite(ptr, size, nitems, stream)
char    *ptr;
int     size, nitems;
FILE    *stream;
```

VOLUME 10 LAB

Write a program for each of the following tasks:

1. Accept a file name interactively and test to see if the file is readable.
2. Accept a file name on the command line and test to see if the file is readable.
3. Same as problem 2, but if the file is readable print the first 3 lines. (Program may assume the file has newline characters.)

Building Block Lab

4. In these lab exercises, you will use programs and program fragments written in previous units to create programs that access the file system via the Standard I/O Library functions.

   a. In Volume 8 Lab, you wrote a program (8.7.c) to prompt the user for various data (such as name and telephone number), to store the data in arrays, and to print it on the terminal.

      Rewrite that program as a function. Instead of writing to the terminal, it should send its output to a disc file, named *telephone*, in some convenient place in the user's file system. (On a UNIX System, this file should reside in *$HOME*.) Be sure to open the file in such a manner that our data entry is added to any previous contents.

      The fields in the file should be separated with a horizontal tab character.

      Modify your menu program to call this telephone file data entry routine .

VOLUME 10 LAB - SOLUTIONS

```
/* 10.1.c */
/* Accepts file name interactively and tests to see if it
 * is readable.
 */

#include   <stdio.h>

main()
{
     char  file_name[256];
     FILE  *fp;

     printf("File name: ");
     scanf("%s",file_name);
     if ((fp = fopen(file_name, "r")) == NULL)
          printf("%s not readable\n", file_name);
     else
          printf("%s is readable\n", file_name);
}
```

```
/* 10.3.c */
/* Accepts many file names on the command line and tests
 * to see if they are readable.
 */

#include   <stdio.h>

main(argc, argv)
int        argc;
char       *argv[];
{
           FILE  *fp;
           int    i;

           if (argc < 2){
               printf("Usage: %s file_name\n",argv[0]);
               exit(1);
               }
           for (i = 1; i < argc ; i++)
               if ((fp = fopen(argv[i], "r")) == NULL)
                   printf("%s not readable\n", argv[i]);
               else
                   printf("%s is readable\n", argv[i]);
}
```

```
/* 10.4a.c (cont.) */

        while ( 1 ) {
            printf("\nEnter Last Name(20 char. max.): ");
            gets (linebuf);
            strncpy (lastname, linebuf, 20);
            printf("\nEnter First Name and M.I.(18 char. max.): ");
            gets (linebuf);
            strncpy (firstname, linebuf, 18);
            printf("\nEnter Phone Number(15 char. max.): ");
            gets (linebuf);
            strncpy (phone, linebuf, 15);
            printf("\nEnter Street address(25 char. max.): ");
            gets (linebuf);
            strncpy (address, linebuf, 25);
            printf("\nEnter City & State(20 char. max.): ");
            gets (linebuf);
            strncpy (city_st, linebuf, 20);
            printf("\nZip(10 char. max.): ");
            gets (linebuf);
            strncpy (zip, linebuf, 10);

            fprintf(wptr, "%s\t%s\t%s\t%s\t%s\t%s\t", lastname,
                        firstname, phone, address, city_st, zip);

            printf("\nAnother entry?? (y or n): ");
            gets (linebuf);
            if ( linebuf[0] != 'y' ) break;
            }
}
```

```
/* menu.c  (cont.) */

/* Prints S&L System features menu with user's prompt */
void prtmenu()
{
   printf("\n\n\t\t\tWHAT WOULD YOU LIKE TO DO NOW ?");
   printf("\n\n\t\t1 - Calculate a monthly mortgage payment.");
   printf("\n\n\t\t2 - Generate a mortgage payment schedule.");
   printf("\n\n\t\t3 - Examine a loan account.");
   printf("\n\n\t\t4 - Examine a savings account.");
   printf("\n\n\t\t5 - Look up a telephone number.");
   printf("\n\n\t\t6 - Add telephone numbers to file");
   printf("\n\n\t\t7 - EXIT");
   printf("\n\n\tPlease make your selection (1-6): ");
}

/* Return an integer read from the user's terminal */
/* Flush bad data */
int getselect()
{
        int c, i;
        if ( scanf("%d", &i) != 1 )
            while (( c = getchar()) != '\n')
                        /* flush stdin */

        return (i);
}
```

```
/* 10.4b.c (cont.) */

intyr /= 100.;
intmo  = intyr / 12.;
npmts  = nyears * 12;
pmt = bal * (intmo / (1. - pow(1. + intmo, -(double)npmts)));

prhdng();
printf("%8s %10s %10s %10s %10.2f\n",
     "", "", "", "", bal);

for (i = 1; i <= npmts; ++i) {   /* Print scheduled */
    intpmt = bal * intmo;       /* payment data */
    if (i < npmts)
        prinpmt = pmt - intpmt;
    else
        prinpmt = bal;
    bal -= prinpmt;
    pmttot += intpmt+prinpmt, intpmttot += intpmt;
    prinpmttot += prinpmt;
    printf("%8d %10.2f %10.2f %10.2f %10.2f\n",
        i, intpmt + prinpmt, intpmt, prinpmt, bal);
    if ( !(i % 48) ) {          /* Every 48 lines */
        putchar('\f');          /* Start new page */
        prhdng();
        }
    }
 printf("%8s %10s %10s %10s\n",         /* Print Totals */
     "","==========","==========","==========");
     printf("%8s %10.2f %10.2f %10.2f\n",
         "", pmttot, intpmttot, prinpmttot);
}

/* Print page headings */
void prhdng()
{
static char ul[] = "**********";
printf("%8s %10s %10s %10s %10s\n",
    "payment", "total", "interest", "principal", "balance");
printf("%8s %10s %10s %10s %10s\n",
    "number", "payment", "payment", "payment");
printf("%8.8s %10s %10s %10s %10s\n", /* Underline Headings */
    ul, ul, ul, ul, ul);
}
```

```
/* 10.4c.c  (cont.) */

while       ( 1 ) {
   printf("\nEnter Last Name(20 char. max.): ");
   gets (linebuf);
   strncpy (record.lastname, linebuf, 20);
   printf("\nEnter First Name and M.I.(18 char. max.): ");
   gets (linebuf);
   strncpy (record.first_mi, linebuf, 18);
   printf("\nEnter Phone Number(15 char. max.): ");
   gets (linebuf);
   strncpy (record.phone, linebuf, 15);
   printf("\nEnter Account Number(15 char. max.): ");
   scanf("%hd", &record.acct_num);
   printf("\nAVT employee (e) or Public (p) Enter e or p : ");
   scanf("%s", linebuf);
   fflush(stdin);
   record.type = linebuf[0];

   if (record.type == 'p') {
     printf("\nEnter     Street address(25 char. max.): ");
     gets      (linebuf);
     strncpy  (record.address.mail_add.st_addr, linebuf, 25);
     printf("\nEnter     City & State(20 char. max.): ");
     gets      (linebuf);
     strncpy  (record.address.mail_add.city_st, linebuf, 20);
     printf("\nZip(10    char. max.): ");
     gets      (linebuf);
     strncpy  (record.address.mail_add.zip, linebuf, 10);
     }
else {
     printf("\nEnter    Department (10 char. max.): ");
     gets      (linebuf);
     strncpy  (record.address.co_addr.dept, linebuf, 10);
     printf("\nEnter    location (20 char. max.): ");
     gets      (linebuf);
     strncpy  (record.address.co_addr.location, linebuf, 20);
     }
```

```
/* 10.4d.c */
/* Data file name to be supplied as first command line argument */
/* Diagnostics - Exit code 1 - No argument supplied */
/* Diagnostics - Exit code 2 - Failed fopen() */

#include <stdio.h>
#include "acct_rec.h"

struct acct_rec record;

main (argc, argv)
int argc;
char *argv[];
{
        FILE    *fp;
        double  pd_total = 0.;
        int     i;

        if (argc < 2){
            fprintf(stderr, "Usage: %s file_name\n",argv[0]);
            exit(1);
            }
        if ((fp = fopen(argv[1], "r")) == NULL) {
            fprintf(stderr, "Cannot open: %s\n", argv[1]);
            exit(2);
            }

        while (1) {
             fread(&record, sizeof(record), 1, fp);
             if (feof(fp)) {
                 fclose(fp);
                 printf("Past Due Total = $%.2f\n", pd_total);
                 exit(0);
                 }
             for ( i=0 ; i < record.num_loans; ++i ) {
                  pd_total += record.loans[i].past_due;
                  if ( record.loans[i].past_due > .01 )
                      printf("%s %d %.2f\n", record.lastname,
                           record.acct_num,
                           record.loans[i].past_due);

        }
    }
;
```

VOLUME 11

Advanced Pointer Use & Special Topics

POINTERS VS ARRAYS

- Declaration

```
char   name[35];   /* storage for 35 bytes */

char   *p;         /* storage for 1 address */
```

- Assignment

```
p = name;      /* Pointer initialization */
               /* required before */
               /* p can be used */

name = address         /* Illegal */
```

- The [ ] and * operators may be used with both arrays and pointers.
  - Convention to use

```
[ ] with arrays
      name[i] = 's';

* with pointers
      p = &name[i];
     *p = 's';
```

ARRAY OF POINTERS

- An array of pointers is an array of addresses
- Declaration:

type **identifier*[*integer-expression*];

EXAMPLE:

Assume the array *buffer* has been filled with text. The function number_line() will record the start address of each line in the array of character pointers *line_num*, e.g.,

line_num[0] will point to the first line
line_num[1] will point to the second line
etc...

The list of line pointers is terminated with a NULL.

```
 5      char  buffer[30001];  /* Null-terminated text buffer */
 6      char  *line_num[3000];          /* Max 3000 lines */
        ...
53      number_lines()
54      {
55              int  i;
56              char  *p;
57
58              line_num[0] = buffer;
59              for ( p = buffer, i = 1; *p != NULL; p++)
60                      if (*p == '\n')
61                              if (*(p + 1) != NULL)
62                                      line_num[i++] = p + 1;
63              line_num[i] = NULL;
64      }
```

DOUBLE POINTERS

This is another version of a scanner. This scanner implements a double pointer in the is_keyword() function.

```
1       /* A sample scanner */
2       /* identifies commands */
3
4       #include        <stdio.h>
5       #include        <string.h>
6       char    *keyword[] = {"append",
7               "find",
8               "list",
9               "remove",
10              "relace",
11              "substitute",
12              ""};
13
14      /* Returns index of command, else -1 */
15      is_keyword(str)
16      char *str;
17      {
18              char    **p;
19              int i = 0;
20              for ( p = keyword; *p != (char *)NULL;p++, i++ )
21                      if (strcmp(str, *p) == 0)
22                              return(i);
23              return(-1);
24      }
```

argv REVISITED

```
1       /* Prints command line arguments */
2       /* using argv as a double pointer */
3       #include        <stdio.h>
4
5       main(argc, argv)
6       int     argc;
7       char    **argv;
8       {
9               for ( ; *argv != (char *)NULL; argv++)
10                      printf("%s"\n", *argv);
11      }
```

```
$ a.out file1 file2
a.out
file1
file2
```

Hypothetical Stack

addr	
	...
1088	2000
	...
2000	2020
2004	2025
2008	2031
2012	0
	...
2020	a.out\0
2026	file1\0
2032	file1\0
	...

VOLUME 11 LAB

1. Examine the first character of each of the command line arguments. If it is a minus sign '-', print the remaining characters of the argument followed by the string " option selected".

2. Write two programs that each will do the following tasks. Write the first treating *argv* as an array of pointers. The second should use double pointer notation.

   a. Print the execution name of the program.

   b. Print the number of parameters passed to it.

   c. Print the parameters that have two or more vowels.

3. **Dynamic Memory Allocation - Advanced Lab Problem**

   Utility programs, such as editors and sorters, often use a technique similar to this lab exercise to gather their input. Write a program that inserts lines entered at the keyboard into a singly linked list. Input is terminated when a period is entered at the beginning of a line. Then all the lines in the linked list are displayed.

   (*Hints:* Declare a character array large enough to hold your longest expected line of input. Use **gets()** to read in a line, followed by **strlen()** to determine its length. Remember that **strlen()** does not count the null character terminating the string. After determining the line length, use **malloc()** to allocate the needed storage, linking the new entry into a list of earlier entries.)

VOLUME 11 LAB - SOLUTIONS

```
/* 11.2.array.c */
/* Print arguments having two or more vowels */
/* Using array of pointer notation */

main(argc,argv)
int argc;
char *argv[];
{
   int k = 1;
   int x;
   char *p;

   printf("The name if the program is %s\n",*argv);
   printf("The number of parameters passed is %d\n",argc);

   while( --argc  ) {
   x = 0;
   p = *(argv+k);
   while( *p != 0 )
         switch( *p++ )
             {
             case 'a':
             case 'e':
             case 'i':
             case 'o':
             case 'u':
                 x++;
             }

     if( x > 1 )
        printf("The parameter %s has %d vowels\n",
            *(argv+k), x);
     k++;
   }
}
```

VOLUME 11 LAB - SOLUTIONS

```
/* 11.3.c
 * This program illustrates a singly linked list that is
 * dynamically growing.
 */

#include <stdio.h>
#include <string.h>

struct node {
            char *data;
            struct node *next;
};

main(argc, argv)
int argc;
char *argv[ ];
{
   char line[BUFSIZ], *malloc();
   struct node *head, *here, *p, *create();

   /* initialize the linked list with a dummy node */
   head = (struct node *) malloc(sizeof(struct node));
   head->next = (struct node *) NULL;
   here = head;

   /* get a line of input and append it to the linked list */
   printf("Enter lines of text:\n");
   while (gets(line) != (char *) NULL) {
               if (line[0] == '.')
                   break;
               here->next = create(line);
               here = here->next;
   }
   /* print out the lines in the linked list */
   for (p = head->next; p != (struct node *) NULL; p = p->next)
            puts(p->data);
}
```

VOLUME 12

Style and Debugging

Appendix

DEBUGGING TOOLS

NAME	USAGE	DESCRIPTION
list(1) and dis(1)		
	cc -g prog.c list prog.c dis a.out	'cc -g' puts C-source line number info in the a.out file. dis(1) disassembles the a.out, giving an assembly language listing with C-source line numbers. Use list(1) to cross reference line numbers that appear in the dis(1) output.
adb(1)	adb	DEC only; absolute debugger; predecessor of sdb(1) and more difficult to use.

sdb QUICK REFERENCE SHEET

```
=====================================================
BREAKPOINTS and TRACES
     SET                              DELETE
   *func:b        first executable statement in "func".    ¦ *func:d
   *12b         source-file-line 12              ¦ *12d
   *func:12b    (Example in above line is sufficient)    ¦ *func:12d
   *b         set brkpnt at current line.
   *B         print all breakpoints.
   *d         delete breakpoints interactively.
   *D         delete all breakpoints.
   *12 b t;x/     at 12,do trace; print the value of x; don't stop running.
   *func:a        at func, print func name and args; don't stop running.
   *func:12a    at 12, print source file line; don't stop running.
=====================================================
RUN THE PROGRAM
   *r  *r args *R    run it with previous args, new args, no args.
   *R         run it without args.
   *r args          run it with these args.
   *c         continue running (after it reached a breakpoint).
   *proc:12c    continue till you get to this line (temporary brkpnt).
   *C         continue prog;pass signal which stopped it back to prog.
   *17g         resume execution at line 17 in curnt func. Skip bad code.
   *s         single step    *S  single step w/o visiting called funcs
     *func(arg1,arg2,....)     execute a function from within sdb.
     *func(arg1,arg2,....)/     execute a function and print the return value.
   *k         kill process from a breakpoint.
   *l         print last line executed.
=====================================================
OUT     *q         quit sdb.
   *!cmd         escape to the shell.
=====================================================
MACHINE LANGUAGE DEBUGGING - use ? to get machine language instruction
              use : for absolute address
   *0x1024?     mach-lang-instr of 0x1024 in crnt function.
   *0x1024:?     mach-lang-instr of absolute address 0x1024
   *CTRL D          see next 10 instructions.
   *0x1024:b     set breakpnt at absolute addr 0x1024.
  LOOK AT MACH CODE:   *0:?          *CTRL D   for next window
=====================================================
```

EFFICIENCY CONSIDERATIONS

```
/*************************************************/

   /*    'C' LANGUAGE -- EFFICIENCY     */
```

1. Use pointers.

2. Put heavily used variables (ints and pointers) into registers.

```
   register    i;

   register    char *p;
```

3. Test for 0.

```
   while (x != 0)          is faster than          while (x < y)

   for (x = MAX; x; x-- )  is faster than   for (x = 0; x < MAX ; x++ )
```

4. Macros are faster than functions (however, macros make executable files larger and can be difficult to debug).

5. UNIX SYSTEM TOOL FOR OPTIMIZING C PROGRAMS: prof(1)

```
   USAGE:    $ cc -p prog.c     /* This creates mon.out */

             $ a.out            /* Run your program. */

             $ prof
```

prof(1) will generate a report telling how much time is spent in each function, the number of times a function is called, etc. By observing where the bottlenecks are and optimizing those functions, you can speed up your program significantly.

EFFICIENCY AND PORTABILITY TIPS

```
/*******************************************************/
   /*    'C' LANGUAGE -- PORTABILITY     */

   1. lint -p prog.c     /* Should come through clean. */

   2. pcc(1)             /* Portable C compiler (PDP only)

   3. Use short or long instead of int.

   4. Be aware that when shifting right, some machines do
      an arithmetic shift, others a logical shift.

   5. Some machines will do sign extension, others won't;
      e.g., char c;
            c = -1;
            if ( c == -1)   /* 3B: not true (sign not extended) */
                            /* DEC: true (sign extended) */


   6. Byte Ordering:
       Different machines store the bytes that make up
       an integer in different orders.

   7. For I/O, use stdio(3S) functions (buffered I/O), not
      system calls (Section 2).
```

OVERVIEW OF scanf()

scanf()

NAME
scanf

SYNOPSIS
int scanf(format[,pointer list])

DESCRIPTION
Reads characters from *stdin*.
format - %d, %f, %s, etc.
Stores results at addr's in pointer list.
Returns # of %'s matched, -1 on EOF.

WARNING
Every argument to scanf() must be an address.

```
main( )
{
    int   num, ret;

    printf("Please enter an integer: ");
    ret = scanf("%d", &num);
    if (ret == 1)
       printf("Thank you\n");
    else
     /* take corrective action */
    .
    .
```

scanf() CONVERSION CHARACTERS (PARTIAL LIST)

%c	any 1 char
%d, u, o, x, X	Precede with l (el) if long, h if short
%e, f, g	Precede with l (el) if double
%s	string of non-whites NULL added to destination

```
main()
{
    int     num, ret;
    float   f;
    charstr  [100];

    ret = scanf("%d %f %s",&num, &f, &str[0]);
    if (ret != 3)
        /* take corrective action */
    .
    .
```

HOW scanf() WORKS

- Reads from *stdin*
- Skips leading white space (except %c)*
- Stops on first conflict, offending char left unread
- Return value: # of %'s matched, -1 on EOF

Example 1:

1	2	3	P	a	t		4	5	6	\n	

```
ret = scanf("%d%s%d",&int1,&str[0],&int2);
```

Return code?
Next char to be read?

Example 2:

7	3	.	2	b	e	e	p	\n			

```
x = scanf("%f %d",&float1, &int1);
```

Return code?
Next char to be read?

* Also [scanset] does not skip leading whites

WHEN scanf() FAILS

This attempt to ensure valid input does not work. When scanf() fails, the "offending character" (in this case 'W') is left on the input stream. Subsequent scanf()'s will also "see" this 'W' and fail. The programmer must clear this bad input from the input stream. The following pages address this problem.

```
/* Attempt to force user to enter valid input */

main()
{
    int    age;

    printf("Enter age: ");
    while (scanf("%d",&age) != 1)
         printf("Try again.  Age: ");
    .
    .
}
```

Terminal Screen:

```
Enter age: Why?
Try again.  Age: 42
Try again.  Age: 42
Try again.  Age: Listen to me!
Try again.  Age: <DELETE>
```

Input:

W	h	y	?	\n	4	2	\n	4	2	\n	L	...

scanf() - ERROR RECOVERY

- Clear to end of line, field, or record
- Or exit the program

```
1  main()
2  {
3     int    age;
4     printf("Enter age: ");
5     while (scanf("%d",&age) != 1) {
6           while ((c = getchar()) != '\n')
7                 ;       /* Clear line */
8           printf("Try again.  Age: \n");
9           }
10 printf("Thank you\n");
11 .
```

Terminal Screen:
Enter age: *Why?*
Try again. Age: *42*
Thank you.

Input:

W	h	y	?	\n	4	2	\n

scanf() - ADDITIONAL FORMATTING

Skip field %*d, %*f, %*s, etc.

C	A		2	2		7	4	7	\n

```
x = scanf("%s %*d %d",&state[0],&val);
```

Literal characters

3	5	2	:	2	9	:	9	9	\n

```
ret_code = scanf("%d:%d:%d",&f1,&f2,&f3);
```

White space

Skips to next non-white

Sometimes useful before %c

5	7		+		4	9	\n

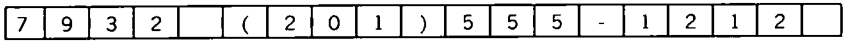
```
x = scanf("%d%c%d",&num1,&op,&num2);
```

```
x = scanf("%d  %c%d",&num1,&op,&num2);
```

scanf() - MAXIMUM FIELD WIDTH

Common Usage	
%2d	2 digit integer
%25s	Maximum 25 non-whites; appends NULL; prevents overwriting array
%1s	First non-white
%20c	Exactly 20 chars (No NULL appended)

Example 1:




```
x = scanf("%2d%d%25s", &loc,  &dept, &phone[0]);

    /* phone array should be at least 26 bytes */
```

Example 2:


```
x = scanf("%13c%d", &name[0], &id);
```

VOLUME 12 SUMMARY
scanf()

This page summarizes the key points to remember when using scanf().

- Reads characters input from *stdin*
- %c, %d, %f, %s, etc.
- All arguments must be addresses
- Check return code for status
- If scanf() fails:
  - — Clear to end of line, field, or record
  - — Or exit program

scanf() - SAMPLE PROGRAM

```
1  /* Finds volume of sphere given radius */
2
3  main()
4  {
5     double   r, pi = 3.141592654;
6     int      c;
7
8     printf(Radius? ");
9     while (scanf("%lf",&r) != 1) {
10             while ((c = getchar()) != '\n')
11                    ;  /* Clear line */
12             printf("Illegal input.  Radius: ");
13             }
14    printf("Volume is %.2f\n",4.0/3.0*pi*r*r*r);
15 }
```

Terminal Screen:

Radius? *thirty and a half*
Illegal input. Radius? *30.5*
Volume is 118846.87

Index

A

additive operators 2-5
address 1-2
& address operator 1-3
& address operator 1-4
& address operator 1-5
ANSI Standard 1-4
argv 11-11
argv 11-8
argv 11-9
arithmetic data types 1-4
array - copying 3-7
array 3-2
array 3-3
array 3-4
array 3-5
array of pointers 11-5
array of pointers 11-6
array subscripts 3-8
arrays and pointers 11-3
arrow operator -> 1-8
arrow operator -> 1-9
ASCII 1-6
ASCII table 1-6
assignment 2-5
atof() 1-10
atoi() 1-10
atol() 1-10
automatic 6-11

E

efficiency 1-7
EOF 2-11
exp() 1-13
expression 2-4
external 6-11

F

fclose() 10-4
fclose() 10-5
feof() 10-14
feof() 10-15
ferror() 10-14
ferror() 10-15
fflush() 10-12
fgetc() 10-10
fgetc() 10-16
fgetc() 10-8
fgets() 10-10
fgets() 10-16
fgets() 10-8
FILE 10-3
float 1-4
floating point constant 1-5
flushing a buffer 10-12
fopen() 10-3
fopen() 10-4
fopen() 10-5
fopen() 10-6
for loop 3-12
Formatted output 2-9
fprintf() 1-16
fprintf() 10-10
fprintf() 10-8
fputc() 10-10
fputc() 10-16
fputc() 10-8
fputs() 10-10
fputs() 10-16

L

left justify-printf() 1-2
library 1-10
library 1-12
library 1-8
library 1-9
log() 1-13
log10() 1-13
logical negation 2-5
logical operators 2-5
long - constant 1-5
long 1-4
long float 1-4
lvalue: array name 3-7

M

malloc() 11-10
minimum field width-printf() 1-2
multiplicative operators 2-5
mv 2-12

O

object code 1-8
object code 1-9
octal - integer constant 1-5
-> operator 1-8
-> operator 1-9
operator 2-5
Operator Precedence 2-3

P

pointer 11-2
* pointer operator 1-2
* pointer operator 1-3
* pointer operator 1-4

sin() 1-14
size of arithmetic data types 1-4
sqrt() 1-13
square root - sqrt() 1-13
stack 6-6
stack 6-7
Standard C Library 1-10
Standard C Library 1-12
statement 2-4
static 6-11
stdin 12-10
storage class 6-11
strcat() 1-9
strcmp() 1-9
strcpy() 1-9
stream 10-16
stream 10-8
string constant - assignment to character pointer 11-4
string constant 1-5
string functions 1-8
string functions 1-9
string literal 1-5
string(3C) 1-8
strlen() 1-9
strlen() 12-11
strncpy() 1-9
structure declaration 1-2
structure declaration 1-3
structure member referencing, the -> operator 1-8
structure member referencing, the -> operator 1-9
structure member referencing, the dot operator 1-4
structure pointers 1-8
structure pointers 1-9
structure tag 1-2
structure tag 1-3
structure template 1-2
structure template 1-3
subscripts - array 3-8
switch 3-15
symbolic debugger 12-5
symbolic debugger 12-6
symbolic debugger ...1 12-4