F

Answers

Day 1 Answers

Following are answers to the quiz and exercises for Chapter 1.

Quiz Answers

1. As you begin to write bigger programs and develop more advanced programs, you will find that memory management is of vital importance. As early as Day 2 you will see several advanced concepts that require memory to be allocated dynamically. These include variable length structures and linked lists.
2. Dynamically allocating memory means that you allocate memory at runtime. In other words, the computer does not reserve room for the variable until a dynamic memory allocation function is executed.
3. You can allocate all the free memory within a computer.
4. You free it. To free dynamically allocated memory, you use one of the free functions. If the memory was allocated with `malloc()`, `calloc()`, or `realloc()`, the ANSI function `free()` is used. For memory allocated with a `far` function, a `far free` function should be used.
5. The `calloc()` function has an additional parameter. The `calloc()` function enables you to declare a number of instances of a specific size of memory. This could be done with the `malloc()` function by breaking out the parameter as follows:

   ```
   calloc( 10, 15 );  /* allocates 10 sets of 15 */
   malloc( 10*15);    /* allocates 10 * 15 */
   ```

6. The `malloc()` function is limited to 64K or less of total memory that can be allocated (based on what is not already being used for other data). The far function can allocate beyond this 64K limit.

Exercises Answers

1. The following is one of many possible answers:

   ```
   int *numbers;
   numbers = (int *) malloc( 10 * sizeof(int) );
   ```

2. The following is one of many possible answers:

```
int *numbers;
numbers = (int *) calloc( 10, sizeof(int) );
```

3. This could be considered a trick question. This exercise can be answered using either the `far` version of the `malloc()` or `calloc()` functions.

   Borland compiler answers:

```
return_value = (long *) farmalloc( 20000 * sizeof(long) );
return_value = (long *) farcalloc( 20000, sizeof(long) );
```

   Microsoft compiler answers:

```
return_value = (long *) _fmalloc( 20000 * sizeof(long) );
return_value = (long *) _fcalloc( 20000, sizeof(long) );
```

4. This program has two notable problems. The first is that the `malloc()` function needs to be typecast to the type of value it is returning. Because it is returning to the variable string, you can assume `malloc()` is returning a character pointer. The second problem is that the variable, `string`, is not declared. The following is a corrected listing.

```
#include <stdlib.h>
#include <stdio.h>
#define MAX  100
void main(void)
{
    char *string;
    string = (char *) malloc( MAX );
    printf( “Enter something: “ );
    gets( string );
    puts( string );       /* do something like printing */
    free( string );
}
```

5. This program has the same problem as some of the listings that you saw today. This program does not free the memory that it allocates. The following is a corrected listing with the `free()` function added.

```
/* Day 1: Exercise 5 */
#include <stdlib.h>
#include <stdio.h>
```

```
void main(void)
{
   long *long_array;
   long  total = 0;
   int   ctr;

   long_array = calloc( sizeof(long), 10 );
   printf( “Enter 10 numbers: “ );
   for( ctr = 0; ctr < 10; ctr++ )
   {
       scanf( “%ld”, long_array+ctr );
       total += *(long_array+ctr);
   }

   printf( “\n\nTotal of numbers is: %ld”, total );
   free(long_array);
}
```

Day 2 Answers

Following are answers to the quiz and exercises.

Quiz Answers

1. The basic data types are `char`, `int`, `short`, `long`, `float`, and `double`.
2. The `typedef` command.
3. DWORD is a type definition of type `unsigned long`.
4. You can group data in three different ways:
   - ☐ With Arrays
   - ☐ With Structures
   - ☐ With Unions
5. The value of NULL is zero. Displayed as a character, this is ‘\0’.
6. Word alignment determines how information is stored. If word alignment is on, then there may be gaps between data items. When word alignment is on, all variables are aligned on a word boundary. If word alignment is off,

variables will be packed as close as possible. To know how data is stored, you must know the status of word alignment.

7. A pointer is a special kind of variable. A pointer is a numeric variable that is used to hold a memory address.
8. The "address of" operator—the ampersand (&)—is used to determine the address of a variable.
9. The first is a prototype for a function pointer, the second is a prototype for a function that returns a pointer.
10. One of the main benefits of using a variable length structure is storage space. Only the amount of space needed is used. Using standard structures you may have a large amount of wasted (blank) space, or, your structure may be too small to store everything.

Exercises Answers

1. The following is one example of how this exercise can be solved using a structure:

```
struct ssn_tag {
     int first;
     char breaker1;
     int middle;
     char breaker 2;
     int last;
};
```

   The following is an alternate solution:

```
struct ssn_tag {
     char first[3];
     char breaker1;
     char middle[2];
     char breaker 2;
     char last[4];
};
```

2. The following is one example of how this exercise can be solved. To store two different variables at the same memory address, a union is used as follows:

```
union tag {
    long number;
    char string[4];
};
```

3. The following is one possible answer:

```
/* Program:  EXER0203.c
 * Author:   Bradley L. Jones
 * Purpose:  A program that does not do much. A pointer is
 *           assinged its own address as a value.
 *=======================================================*/

#include <stdio.h>

void main(void);       /* prototype for main() */

void main(void)
{
   long *ptr;

   ptr = &ptr;     /* this line may generate a warning */

   printf(“\nThe value of ptr is:   %ld”, ptr);
   printf(“\nThe address of ptr is: %ld”, &ptr);
}
```

Output

```
The value of ptr is:   -720908

The address of ptr is: -720908
```

4. The following is one of many possible answers. This listing includes two different double dimensioned arrays. While not covered in detail in today’s material, a double dimensioned array is simply an array of arrays. Notice that the third printing switches the subscripts and causes erroneous information to be printed.

```
/* Program:  EXER0204.c
 * Author:   Bradley L. Jones
 * Purpose:  Creates a double dimensioned character array
 *           and initializes it.  Also creates a double
 *           dimensioned integer array and initializes it.
 *           Once declared, both arrays are printed out.
 *======================================================*/

#include <stdio.h>

void main( void );         /* prototype for main() */

char checkerboard[8][8] = {‘X’,’O’,’X’,’O’,’X’,’O’,’X’,’O’,
                          ‘O’,’X’,’O’,’X’,’O’,’X’,’O’,’X’,
                          ‘X’,’O’,’X’,’O’,’X’,’O’,’X’,’O’,
                          ‘O’,’X’,’O’,’X’,’O’,’X’,’O’,’X’,
                          ‘X’,’O’,’X’,’O’,’X’,’O’,’X’,’O’,
                          ‘O’,’X’,’O’,’X’,’O’,’X’,’O’,’X’,
                          ‘X’,’O’,’X’,’O’,’X’,’O’,’X’,’O’,
                          ‘O’,’X’,’O’,’X’,’O’,’X’,’O’,’X’};

int number_array[3][4] = { 11, 12, 13, 14,
                           21, 22, 23, 24,
                           31, 32, 33, 34 };

void main( void )
{
   int x, y;               /* counters */

   printf( “\n\nPrint Character array...\n”);

   for (x = 0; x < 8; x++)
   {
       printf( “\n” );
       for ( y = 0; y < 8; y++ )
       {
           printf( “%c “, checkerboard[x][y] );
       }
   }

   printf( “\n\nPrint numbers with subscripts...\n”);

   for (x = 0; x < 3; x++)
   {
        printf( “\n” );
        for ( y = 0; y < 4; y++ )
        {
            printf( “%d “, number_array[x][y] );
        }
    }

printf( “\n\nPrint numbers with switched subscripts..\n”);

    for (x = 0; x < 4; x++)
    {
        printf( “\n” );
        for ( y = 0; y < 3; y++ )
        {
            printf( “%d “, number_array[x][y] );
        }
    }
}
/*** end of listing ***/
```

Output

```
   Print Character array...
X O X O X O X O
O X O X O X O X
X O X O X O X O
O X O X O X O X
X O X O X O X O
O X O X O X O X
X O X O X O X O
O X O X O X O X

   Print numbers with subscripts...
11 12 13 14
21 22 23 24
31 32 33 34

   Print numbers with switched subscripts...
11 12 13
21 22 23
31 32 33
2570 29264 28265
```

5. You would create an array of the social security number unions:

```
struct ssn_tag SSN[10];
```

6. The numeric array can be printed as characters by using `"%c"` and printing each element in a `printf()` statement. The numbers print “BRADLEY”.
7. This exercise is “On Your Own.”
8. This exercise is “On Your Own.”
9. This exercise is “On Your Own.”

Day 3 Answers

Following are answers to the quiz and exercises.

Quiz Answers

1. 0
2. The list is empty.
3. A pointer to the same data type as the linked structure is used.
4. A stack is a single-linked list that is always accessed from the top. New elements are added to the top, and removed elements are always taken from the top. A stack has a LIFO (Last In First Out) access order.
5. A queue is a single-linked list that has new elements added to the top. Elements that are removed from the queue are always taken from the bottom (or tail). A queue has a FIFO (First In First Out) access order.
6. A tail pointer is a separate pointer that always points to the last element in a linked list. A top pointer is a separate pointer that always points to the first element in a linked list. If the linked list is empty, then the top and tail pointers will both contain NULL.
7. A tail pointer is generally not needed for a single-linked list. In some cases, they are needed. For example, a queue needs a tail in order to know where to remove elements from.
8. A double-linked list has a pointer to the previous element. This is in addition to the pointer to the next element that both a doubly and single-linked list have. Because there is a pointer to the previous element, a double-linked list can be traversed either forward or backward.

9. `calloc()` also initializes the new element to zero. `malloc()` does not initialize.
10. A binary tree has a much quicker access time when looking for specific elements.

Exercises Answers

1. Following is one of many possible solutions:

   ```
   struct character {
       char character_name[25+1];
       int  introduction_year;
       struct character *next_character;
   };
   ```

2. Following is one of many possible solutions:

   ```
   struct food_item {
       char food_name[25+1];
       struct food_item *next_food;
       struct food_item *previous_food;
   };
   ```

3. Following is one of many possible solutions:

   ```
   struct member {
       char name[25+1];
       int age;
       struct member *left_node;
       struct member *right_node;
   };
   ```

4. This is on your own.
5. The structure should contain a pointer to the next customer, not another structure. The corrected structure is:

   ```
   struct customer {
       char lastname[20+1];
       char firstname[15+1];
       char middle_init;
       struct customer *next;
   };
   ```

6. D B E A L I M G N J F K H C
7. A B D E C F G I L M J N H K
8. D E B L M I N J G K H F C A
9. This exercise is on your own.

Day 4 Answers

Following are answers to the quiz and exercises.

Quiz Answers

1. American National Standards Institute
2. Because ANSI sets the standards for many areas. This includes the use of programming languages such as C and C++ along with the use of terminals and more.
3. The ANSI functions are easy to use.
4. The ANSI functions require that the ANSI.SYS driver be loaded on the computer before they will work properly. If this driver is not loaded, then erroneous output will most likely be received.
5. This is a trick question! Red has two values. The foreground color is 31, and the background color is 41.
6. The foreground color is the color applied to the text or graphic being displayed. The background color is the color behind the text or graphic being displayed.
7. When the next ANSI color is applied.
8. It is the fastest method of updating the screen.
9. It is a less compatible means of updating the screen. This is because video memory may be located in different areas of RAM on different machines.
10. Basic Input/Output System

Exercises Answers

1. The escape sequence is an ANSI escape sequence. If the ANSI.SYS driver is loaded, then this will move the cursor up five lines (or as many as possible if there are not five preceding the cursor's original position).

2. Black (30) on White (47)

   White (37) on Black (40)

   Yellow (33) on Blue (44)

   Yellow (33) on Red (41)

3. Yellow on Blue should have been easy. The Bright Yellow may have confused you. Bright Yellow is the same as bold yellow. You gain a bold color by adding a 1 to the escape sequence.

   Yellow on Blue `“\x1B[33;44m”`

   Bright Yellow on Red `“\x1B[1;33;41m”`

4. The escape sequence uses two foreground colors. The first foreground color will be ignored. Typically you will use a foreground and a background color.

5. This function uses functions provided in today's lessons. This function works just fine if all of the functions used are included. Be careful with screen coordinates. Some programs (and functions) use 0,0 as the starting point, others use 1,1.

6. The following is one possible answer. A `main()` has been added to demonstrate the use of the function. Line numbers have been included. Note that this program must be linked with a_cursor.obj presented in today's chapter.

```
/* Program: EXER0406.c
 * Author:  Bradley L. Jones
 * Purpose: demonstrate the put_color_string() function
 *          This program puts the word “Hello” on the
 *          screen in a bunch of colors.
 *=======================================================*/

#include <stdio.h>
#include “a_cursor.h”
#include “ansiclrs.h”

```

```
/*----------------------
    Function prototypes
  ----------------------*/

void put_color_string( int row, int col,
          int fcolor, int bcolor,
          char *text );
void set_color( int fore, int back );
void main(void);

void main(void)
{
   int row, col,
       x, y;

   set_color( F_WHITE, B_MAGENTA );

   clear_screen();

   for( y = 40; y <= 47; y++ )
   {
      col = ( y - 39 ) * 7;

      for( x = 30; x <= 37; x++ )
      {
         row = x - 29;
         put_color_string( row, col, x, y, “Hello” );
      }
   }
   set_color(F_WHITE, B_BLACK );  /* your screen color */
}

/*------------------------------------------------------*
 * Function:    put_color_string()
 *
 * Parameters:  row, col       - Screen coordinates.
 *              fcolor, bcolor - String colors
 *              text           - The string
 *
 * Notes:   This function does not verify valid values
 *          for the parameters received.
 *------------------------------------------------------*/

void put_color_string( int row,    int col,
                       int fcolor, int bcolor, char *text )
{
  put_cursor( row, col );
  set_color( fcolor, bcolor );
  printf( text );
}

void set_color( int fore, int back )
{
   printf(“\x1B[%d;%dm”, fore, back );
}
```

7. The following is one possible answer. Line numbers have also be included.

```
/* Program: EXER0407.C
 * Author:  Bradley L. Jones
 * Purpose: Re-maps the SHIFTed Function keys to new
 *          values.  After this program is ran without
 *          any command line parameters, the following
 *          are set:
 *
 *          <shift> F1 = HELP
 *          <shift> F2 = DIR
 *          <shift> F3 = CLS
 *          <shift> F4 = CHKDSK
 *          <shift> F5 = CD C:\ID\T7G
 *
 * Note:    Running the program with an extra command
 *          line parameter will reset the keys.
 * Note:    The ‘13’ is <enter> - These commands start
 *          automatically.
 *=======================================================*/

```

```
#include <stdio.h>

#define SHIFT_F1  “0;84”
#define SHIFT_F2  “0;85”
#define SHIFT_F3  “0;86”
#define SHIFT_F4  “0;87”
#define SHIFT_F5  “0;88”

int main(int argc)
{
    if( argc < 2 )
    {
        printf(“\x1B[%s;72;69;76;80;13p”, SHIFT_F1 );
        printf(“\x1B[%s;68;73;82;13p”, SHIFT_F2 );
        printf(“\x1B[%s;67;76;83;13p”, SHIFT_F3 );
        printf(“\x1B[%s;67;72;75;68;83;75;13p”, SHIFT_F4 );
        printf(“\x1B[%s;77;69;77;13p”, SHIFT_F5 );

        printf(“\n\nThe Function keys have been set for “);
        printf(“shift commands.\n\n”);
    }
    else
    {
        printf(“\x1B[%s;%sp”,SHIFT_F1, SHIFT_F1);
        printf(“\x1B[%s;%sp”,SHIFT_F2, SHIFT_F2);
        printf(“\x1B[%s;%sp”,SHIFT_F3, SHIFT_F3);
        printf(“\x1B[%s;%sp”,SHIFT_F4, SHIFT_F4);
        printf(“\x1B[%s;%sp”,SHIFT_F5, SHIFT_F5);

        printf(“\n\nThe Function keys have been reset.\n\n”);
    }

    return;
}
```

Exercises 8 through 10 are on your own.

Day 5 Answers

Following are answers to the quiz and exercises.

Quiz Answers

1. It depends on the overall requirements of the program you are creating. Most times, maintainability will be more important. Efficiency is only important with a program that is time-critical. The most costly part of a program is the time the programmer spends maintaining it.
2. If you are using the ASCII character table, it is 97; however, it could be any other value. A character's numeric value is based on the character set used.
3. The largest unsigned character value will be defined within the numeric constant `UCHAR_MAX`.
4. American National Standards Institute.
5. Most C compilers are case sensitive. For most compilers, each of the variables would be treated as different. These variable names would not be portable since some compilers don't differentiate case.
6. The `isalpha()` function is an ANSI function that determines if a character is a letter of the alphabet. It's better to use this function than a function that checks to see if the character is between a and z.
7. The `isdigit()` function checks to see if a character is a number (0, 1, 2, 3, 4, 5, 6, 7, 8, and 9). This is an ANSI function. It's better to use `isdigit()` than to check to see if a character is greater than or equal to 0 and less than or equal to 9.
8. As stated in the answers to quiz questions 6 and 7, the functions are ANSI compatible. Using these functions will produce portable code. Because there isn't a proprietary character set for the C language, there is no guarantee what numeric values will represent characters. By using the `is...()` functions, you gain portability.
9. No. Generally, when working with data files and structures, you must ensure that all things are consistent. Issues such as byte alignment must be known. Because one compiler may have byte alignment on and another may not, portability may be lost.
10. No. The predefined constants are replaced at compile time, not run time. `__TIME__` will be replaced with the time the program was compiled.

Exercises Answers

1. This program uses `ctr` and `CTR`. These are two separate variables if your compiler is case-sensitive. If you are considering portability, you should rename one of these two variables to allow something other than case to differentiate them.

2. Any listing in this chapter can be used for this exercise. The listing size will shrink; however, the object file and the executable file should remain the same.

   This exercise shows that nothing is added to the program if you add additional spacing. The additional spacing does, however, make the program much more readable.

3. Following is a function that verifies that a character is a vowel.

```
/* Function to verify vowel. All checks are equality, so code will
be portable.*/

int verify_vowel( int ch )
{
    int rv = 0;
    switch ( ch )
    {
        case ‘a’:
        case ‘e’:
        case ‘i’:
        case ‘o’:
        case ‘u’:
        case ‘A’:
        case ‘E’:
        case ‘I’:
        case ‘O’:
        case ‘U’:  rv = 1;
                   break;
       default:    rv = 0
                   break;
    }
    return( rv );
}
```

4. One of many possible answers:

```
int character_case( int ch )
{
    int rv = 0;
    if ( isalpha( ch ) )
    {
        if( isupper( ch ) )
            rv = 1;
        else
            rv = 2;
    }
    else
        rv = 0;

    return( rv );
}
```

5. This exercise was on your own. This information can be found within your compiler's manuals.
6. No. This program uses `system()`. This function calls an operating specific command—in this case `TYPE`. This means that your program isn't portable to other operating systems.
7. No. This is quite similar to a program shown in today's materials. The following text would be much more portable:

```
int to_upper( int x )
{
    if( isalpha( x ) && islower( x ) )
    {
         toupper( x );
    }
    return( x );
}
```

Day 6 Answers

Following are answers to the quiz and exercises.

Quiz Answers

1. Because a computer works with numbers. All characters and symbols are converted to numeric values.
2. For IBM Compatible PCs, the ASCII Standard specifies the conversion values.
3. A decimal value is a base 10 number. Decimal numbers are the numbers that we use to count with every day. Because decimal stands for 10, only 10 numbers are used. These numbers are 0 through 9.
4. The computer stores information in bits. A bit can have one of two states, on or off. Binary numbers are base 2. This means that they can use two digits, 0 and 1, to represent all numbers. These two digits can be used to represent the on and off state of the computer's information.
5. A computer stores information in bytes. A byte is 8 bits. To look at the binary representation of a byte of data means looking at eight digits for a maximum value of 256 (decimal). By using hexadecimal, these 256 possible numbers can be represented in two digits. In addition, the two hexadecimal digits can easily be converted to binary since the first digit represents the left four binary digits and the second hexadecimal digit represents the right four binary digits.
6. While there could be many reasons for wanting to look at the numeric values of data, there is one reason that is most often given. Many characters appear as spaces when viewed as common text. In many cases, the "spaces" may be something quite different. For example, in many of the listings today, you saw the carriage return, line feed. In addition, you saw that spaces were given a decimal 32 value. Other characters such a null (character 0) may also appear as a space when viewed as text; however, 0 and 32 are different.
7. Binary: 0 and 1
8. Decimal: 0, 1, 2, 3, 4, 5, 6, 7, 8, and 9
9. Octal: 0, 1, 2, 3, 4, 5, 6, and 7
10. Hexadecimal: 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, A, B, C, D, E, and F

Exercises Answers

1. Binary: 0100010

   Octal: 102

   Decimal: 66

   Hexadecimal: 42

2. Binary: 10

   Octal: 2

   Decimal: 2

   Hexadecimal: 2

3. a. 88

   b. 32

   c. 120

   d. 49

   e. 3

4. a. A

   b. ? (question mark)

   c. ♣ clover

   d. }

   e. 9

5.
   ```
   /* Program:  Day06E05.c
    * Author:   Bradley L. Jones
    * Purpose:  Print numeric values of an entered character.
    *========================================================*/

   #include <stdio.h>
   #include <stdlib.h>

   char *char_to_binary( int );
   void main(void)
   {
   ```

```
   int ch;
   char *rv;

   printf(“\n\nEnter a number ==>” );
   scanf(“%d”, &ch);

   printf(“\n\n Your number:       %d”, ch );
   printf(“\n\n Character:         %c”, ch );
   printf(“\n Octal value:        %o”, ch);
   printf(“\n Hexidecimal value:  %x”, ch );

   rv = char_to_binary(ch);

   printf(“\n Binary value:       %s”, rv);
   printf(“\n\nYour number again:   %d”, ch );
}

char *char_to_binary( int ch )
{
  int  ctr;
  char *binary_string;
  int  bitstatus;

  binary_string = (char*) malloc( 9 * sizeof(char) );

  for( ctr = 0; ctr < 8; ctr++)
  {
     switch( ctr )
     {
       case 0:  bitstatus = ch & 128;
                break;
       case 1:  bitstatus = ch & 64;
                break;
       case 2:  bitstatus = ch & 32;
                break;
       case 3:  bitstatus = ch & 16;
                break;
       case 4:  bitstatus = ch & 8;
                break;
       case 5:  bitstatus = ch & 4;
                break;
       case 6:  bitstatus = ch & 2;
                break;
       case 7:  bitstatus = ch & 1;
                break;
     }

     binary_string[ctr] = (bitstatus) ? ‘1’ : ‘0’;
   }

   binary_string[8] = 0;  /* Null Terminate */

   return( binary_string );
}
```

6. The following is a complete program.

```
/* Program:  Day06E06.c
 * Author:   Bradley L. Jones
 * Purpose:  convert the case of a letter.
 *=========================================================*/

#include <stdio.h>

unsigned char switch_case( unsigned char );

void main(void)
{
   unsigned char letter1 = ‘c’,
                 letter2 = ‘Y’,
                 converted1,
                 converted2;

   printf(“\n\nThe first letter is %c”, letter1);
   converted1 = switch_case(letter1);
   printf(“\n\n%c is now %c”, letter1, converted1);

   printf(“\n\nThe second letter is %c”, letter2);
   converted2 = switch_case(letter2);
   printf(“\n\n%c is now %c”, letter2, converted2);
}


unsigned char switch_case( unsigned char ch )
{
  /* if lowercase, make uppercase */

  if( ch >= 97 && ch <= 122 )  /* is letter from a to z */
  {
     ch -= 32;        /* change number by subtracting 32 */
  }
  else
  {
     if( ch >= 65 && ch <= 90 )
     {
        ch += 32;        /* change number by adding 32 */
     }
  }

  return(ch);
}
```

7. **BUG BUSTER:** The letter ‘a’ has a decimal value of 97 while the letter ‘Z’ has a value of 90. Since the conditional portion of the `for` statement is to print `x` while it is less than ‘Z’, nothing will ever print. The uppercase ‘Z’ is less than the lowercase ‘a’. Either print only upper- or lowercase letters, or print from ‘A’ to ‘z’. If you select to print from ‘A’ to ‘z’, then you will get a few extra characters printed.

Day 7 Answers

Following are answers to the quiz and exercises.

Quiz Answers

1. A library is a set of functions that have been grouped together. A library allows all of the functions to be grouped into a single file.

2. Many libraries will come with a compiler. Additional libraries can be created by you or others. In addition to your own libraries, you can purchase special purpose libraries.
3. The addition operator (+) is used in a manner similar to the following:

   ```
   LIB +objfile
   ```

4. The subtraction operator (-) is used in a manner similar to the following:

   ```
   LIB -objfile
   ```

5. You combine the subtraction and addition operators are used in a manner similar to the following:

   ```
   LIB -+objfile
   ```

   or you can do a subtraction followed by an addition:

   ```
   LIB -objfile
   LIB +objfile
   ```

6. You add a comma followed by the listfile name to the end of the Library command similar to the following:

   ```
   LIB ,listfile.lst
   ```

7. You use the asterisk in the library command similar to the following:

   ```
   LIB *objname
   ```

8. You combine the subtraction operator and the asterisk as follows:

   ```
   LIB -*objname
   ```

   or you can do the two operations separately:

   ```
   LIB *objname
   LIB -objname
   ```

9. How the file is included determines where the compiler looks for the include files. If double quotes are used, the compilers searches the current directory first. If <> brackets are used, then the Library directory is searched first.
10. Source (.C) files should not be included in libraries. Only object (.OBJ) files should be included.

Exercises Answers

Note: Answers may vary for different compilers. For the Microsoft compilers, the answers should end with a semicolon. For Borland's compilers, you will use TLIB instead of LIB.

1. To create the stated library, you would enter the following:

   ```
   LIB STATES +RI +IL +IN
   ```

   (LIB would be replaced by your library program)

2. `LIB STATES ,LISTING.LST`

   (LIB would be replaced by your library program)

3. `LIB STATES +RI +IL +IN ,LISTING.LST`

4. `LIB STATES +KY.OBJ +FL.OBJ`

5. `LIB STATES -KY.OBJ`

6. `LIB STATES -+FL`

7. `LIB STATES *IL`

8. Two listings are included here. First is UPPER.C which contains the edit. Second is a program that uses the upper edit.

   ```
   /* Program:  upper.c
    * Author:   Bradley L. Jones
    * Purpose:  This is an edit to verify that a string
    *           contains all uppercase alpha characters
    *           (A to Z).
    * Returns:  1 = all uppercase characters
    *           0 = not all uppercase
    *=========================================================*/

   #include <stdio.h>

   int is_upper_case( char *string)
   {
     int ctr;
     int rv;

   ```

```
  for( ctr = 0; string[ctr] != 0; ctr++ )
  {
     if( string[ctr] < ‘A’ || string[ctr] > ‘Z’ )
     {
        return(0);
     }
  }

  return(1);
}



```

```
/* Program:  eday0708.c
 * Author:   Bradley L. Jones
 * Purpose:  This is the main() for testing the upper edit.
 * Returns:  nothing
 *=========================================================*/

#include <stdio.h>

int is_upper_case( char *string );

void main(void)
{
   int  rv1,
        rv2;
   char text1[10] = “BADtext”,
        text2[10] = “GOODTEXT”;

   rv1 = is_upper_case(text1);
   rv2 = is_upper_case(text2);

   printf(“\n\nReturn value = %d for text 1, %s”, rv1, text1);
   printf(“\n\nReturn value = %d for text 2, %s”, rv2, text2);
}
```

9. `LIB EDITS +UPPER.OBJ`
10. This is on your own!

11. This is on your own; however, your answer will be similar to Exercise 9's answer!

12. Although not covered in the chapter, compilers typically work from left to right. This means that all source files should be listed before the libraries are listed. By having `source2.obj` after the library, you could encounter problems.

Day 8 Answers

Following are answers to the quiz and exercises.

Quiz Answers

1. Basic Input/Output System

2. Two possible reasons follow. First, the BIOS functions do not require the use of an external driver such as ANSI.SYS. Secondly, the BIOS functions provide more functionality than the ANSI functions.

3. An assumption as to the location of the video information in memory has to be made when you access video memory directly. By using the BIOS functions, you go through BIOS to access the video. Because different systems may store the video memory at different locations, the BIOS functions are more portable.

4. An interrupt number is a number passed to BIOS via DOS that activates a certain function or type of function. Table 8.1 lists several of the types of functions implemented by different interrupts.

5. An interrupt function is a specific operation within an interrupt. An interrupt function is designated by a number just as interrupts are. In addition, interrupt functions are used in association with interrupts. Table 8.2 lists many of the interrupt functions available.

6. Some interrupt functions are broken down even further. These smaller breakdowns are subfunctions. Several subfunctions are listed in Table 8.2. For example, interrupt 16 (0x10h) function 15 (0x0Fh) has several subfunctions that work with the video colors.

7. Interrupt 0x33h functions manipulate the mouse.

8. It hides the mouse pointer.

9. It reboots the computer.
10. No. BIOS functions are only supported on IBM-compatible machines. In addition, older versions of DOS may not support all of the BIOS functions.

Exercises Answers

1. There is no answer to this exercise. You should end up with a library file that you can use in later chapters of this book. The header file that you create should look like the following:

```
/* Program: TYAC.H
 * Authors: Bradley L. Jones
 *          Gregory L. Guntle
 * Purpose: Header file for TYAC library functions
 *=======================================================*/

#ifndef _TYAC_H_
#define _TYAC_H_

/* DOS and BIOS Interrupts */
#define BIOS_VIDEO       0x10

/* BIOS function calls */
#define SET_VIDEO        0x00

/*------------------------*
    Function Prototypes
 *------------------------*/

     /* Gets the current date */
void current_date(int *, int *, int *);

     /* Positions the cursor to row/col */
void cursor(int, int);
     /* Returns info about cursor */
void get_cursor(int *, int *, int *, int *, int *);
     /* Sets the size of the cursor */
void set_cursor_size(int, int);

```

```
          /* clear the keyboard buffer */
void kbwait( void );
          /* determine keyboard hit */
int  keyhit( void );

#endif
```

2. The program requested is simply a specific version of the function presented in the answer for Exercise 3.

3. The following function enables you to scroll the screen up or down. When the screen scrolls up, the text appears to move down. When the screen scrolls down, the text appears to move up.

```
/* Program: SCROLL.C
 * Author:  Bradley L. Jones
 *          Gregory L. Guntle
 * Purpose: Function to scroll the screen.
 *--------------------------------------------------------
 * Parameters:
 *          row, column    starting position on screen
 *                         (0,0)
 *          height, width  portion of screen to scroll
 *          nbr            number of rows to scroll
 *          direction      0x07 is up
 *                         0x08 is down
 *========================================================*/

#include “tyac.h”
#include <dos.h>

void scroll( int row,   int col,
             int width, int height,
             int nbr,   int direction)
{
   union REGS irreg;

    ireg.h.ah = direction;
    ireg.h.al = nbr;
    ireg.h.ch = row;
    ireg.h.cl = col;
   ireg.h.dh = row + height;
   ireg.h.dl = col + width;
   ireg.h.bh = 0;

   int86( 0x10, &ireg, &ireg );
}
```

4. The following is a program does not have much functionality. It simply shows the scrolling of the screen in action.

```
/* Program: EXER0804.C
 * Author:  Bradley L. Jones
 *          Gregory L. Guntle
 * Purpose: Demonstrates the use of the  set_control_break()
 *          function.
 *========================================================*/

#include “tyac.h”
#include <stdio.h>

#define SCROLL_UP  0x07

void delay(void);
void scroll( int row, int col, int width, int height, int nbr,
             int direction);

int main(void)
{
  int x;

  printf( “\n\nStarting program...”);

  for( x = 0; x < 10; x++ )
  {
    printf( “\rScrolling —>%d”, x);
    delay();
    scroll( 0, 0, 80, 25, 2, SCROLL_UP );
  }
  return 0;
}
```

```

void delay(void)
{
    long x;
    for( x = 0; x < 1000000; x++ );
}
```

5. See Day 7 if you need help adding the `scroll()` function to your library. You should also include defined constants in your TYAC.H header file to define `SCROLL_UP` and `SCROLL_DOWN`. Don't forget to include the prototype in TYAC.H also.

6. The `int86()` function is not used properly. You must pass the interrupt registers in and provide for the interrupt registers coming out. The following is how the `int86()` function should be called:

```
int86(0x21, &inregs, &outregs);
```

Day 9 Answers

Following are answers to the quiz and exercises.

Quiz Answers

1. Monochrome means single colored.
2. Text graphics are graphics created using the characters in a character set such as the ASCII character set.
3. The ASCII characters have numeric values from 0 to 255.
4. The extended ASCII characters are those from 128 to 255.
5. When you change a library function, you must remember to update the library. In addition, you should update any information in header files.
6. If you change a header file, you may need to recompile all the functions and programs that use that header file because the change may affect them too.
7. There are several reasons to use defined constants. The reason presented today was readability.
8. Foreground colors are black, blue, green, cyan, red, magenta, brown, white, gray (light black), light blue, light green, light cyan, light red, light magenta, yellow (light brown), and bright white.

Background colors are black, blue, green, cyan, red, magenta, brown, and white.

Exercises Answers

1. There is no answer to this exercise. You should end up with a library file that you can use in later chapters of this book.
2. See the answer for Exercise 3.
3. The following is one possible answer. The mode is captured, information is displayed, and the program pauses. Once the user presses enter, the mode is changed to 40 columns and the new mode information is displayed. Again the program pauses until the user presses enter. When the user presses enter again, the program resets the mode to its original value.

```
/* Program: EXER0903.c
 * Author:  Bradley L. Jones
 *          Gregory L. Guntle
 * Purpose: Demonstrates the use of the set_video() and
 *          get_video() functions.
 *=======================================================*/

#include <stdio.h>
#include “tyac.h”

int main(void)
{
   int cols, vpage, mode, orig_mode;
   char ch;

   printf(“\n\nGetting video information...”);

   get_video( &cols, &mode, &vpage );

   printf(“\n\n   columns = %d”, cols );
   printf(“\n   mode    = %d”, mode   );
   printf(“\n   page    = %d”, vpage  );

   orig_mode = mode;

   printf(“\n\nPress enter to continue...”);
```

```
    ch = getchar();

    set_video( 1 );

    get_video( &cols, &mode, &vpage );

    printf(“\n\n   columns = %d”, cols );
    printf(“\n   mode    = %d”, mode   );
    printf(“\n   page    = %d”, vpage  );

    printf(“\n\nPress enter to continue...”);
    ch = getchar();

    set_video( orig_mode );

    return 0;
}
```

4. This character is a medium grade grid character. It provides a grid on the screen in the different colors.

Day 10 Answers

Following are answers to the quiz and exercises.

Quiz Answers

1. The purpose of `getline()` is to replace `scanf()` on allowing data to be input.
2. `boop()` causes the speaker to beep for a second.
3. `boop()` will enable you to signal when the user makes an error. By having the computer beep, it provides a signal to the user that there is a problem.
4. `pause()` requires the user to press a key, `waitsec()` does not. In addition, `waitsec()` enables you to specify how long you want to pause.
5. The input text color is set by using option 1. To set yellow on red, you use `getline()` as in the following:

   ```
   getline(1, YELLOW, RED, LIGHTYELLOW, RED, 0)
   ```

   This also sets the highlight color to bright yellow on red.

6. The default colors for `getline()` are as follows:

   normal color is WHITE on BLACK

   highlight color is BRIGHTWHITE on BLACK

   underline color is GREEN on BLACK

   “INS” message is YELLOW on BLACK

Exercises Answers

1. There is not an answer to this exercise.
2. The exit keys have not been set up for `getline()`. You should always ensure that you set up the exit keys so that `getline()` knows which keys will exit. Following is a corrected listing:

```
#include <stdio.h>
#include “tyac.h”

int main()
{
    char ch;
    char strng[40];
    char exit_keys[] = {ESC_KEY, TAB, CR_KEY};

     ch = getline(0,0,0,0,0,0,0);
     ch = getline(8,0,40,0,0,0,strng);
     ch = getline(9,0,4,0,0,0,exit_keys);

     write_string(“Last name:”, LIGHTBLUE, BLACK, 10, 8 );
     ch = getline(6,0,10,20,0,20,strng);  /* Get line */

    return 0;
}
```

3. A single answer is provided in Exercise 4.
4. The following is one of many possible answers for Exercises 3 and 4:

```
/* Program:  EXER1003.c
 * Author:   Bradley L. Jones
 *           Gregory L. Guntle
 * Purpose:  Exercise to create a function to get yes/no.
 *=========================================================*/

#include <stdio.h>
#include “tyac.h”

int main()
{
    char ch;
    char answer[2];
    int i;
    int done = FALSE;

    char exit_keys[] = {ESC_KEY, F1, CR_KEY};

    /* set up getline() for program’s use */
    ch = getline(0,0,0,0,0,0,0);
    ch = getline(1,0,YELLOW,RED,BRIGHTWHITE, RED,0);
    ch = getline(2,0,YELLOW,RED,BRIGHTWHITE, RED,0);
    ch = getline(9,0,4,0,0,0,exit_keys);

    /* clear field */
    ch = getline(8,0,2,0,0,0,answer);

    /* draw message and box */

    while( done == FALSE )
    {
       box( 10, 14, 25, 58, SINGLE_BOX, CLR_INS, YELLOW, RED );
       write_string(“Enter (Y)es or (N)o:”, YELLOW, RED, 12, 30 );

       ch = getline(6,0,12,52,0,1,answer);

       if( answer[0] == ‘Y’ || answer[0] == ‘N’ ||
           answer[0] == ‘y’ || answer[0] == ‘n’ ||
           ch == ESC_KEY || ch == F1)
       {
          done = TRUE;
       }
       else
       {
          boop();
       }

    }

     write_string( “You entered “ , BRIGHTWHITE, BLACK, 20, 1 );
     write_string( answer, BRIGHTWHITE, BLACK, 20, 13 );

    return 0;
}
```

5. This exercise is on your own; however, Day 13 presents a solution.

Day 11 Answers

Following are answers to the quiz and exercises.

Quiz Answers

1. There are times when you don’t want the cursor displayed on the screen. If a menu is currently being used, or if you are waiting for the user to press any key before continuing, then the cursor is not needed. If you are doing data entry, then the cursor is needed.
2. If you call `cursor_off()` and then exit your program, your cursor may remain off. You should always turn the cursor back on if you turned it off.
3. Not all compilers provide a clear screen function.
4. Many compilers do not have a function to clear the screen. Borland provides the `clrscr()` function; however, this function does not give you control over the color that the screen is cleared to.
5. There are three different grid patters in the ASCII table. There are several other graphics characters that could be used, but they aren’t grids. Included in the additional characters is a solid block.
6. The ASCII values of the grid characters are 176, 177, and 178. You can see them in Appendix B.

7. The benefit of saving the screen is that you can easily restore it. If you don't save the screen, then you have to attempt to redraw it.
8. Nothing will happen if you don't restore a saved screen; however, you should make sure that you do free the space allocated to hold the screen information.
9. The row and column information is saved so that the restore function doesn't need to know where the saved screen area goes.
10. Two bytes are allocated for each screen position. One is for the actual character that is displayed. The other is for the attributes such as the color.

Exercises Answers

1. There is no answer to this exercise. The following are additions to the TYAC.H header file:

```
#define READ_CHAR_ATTR    0x08

#define SCREEN_ROW    80*2

     /* Hide/show cursor */
void cursor_off(void);
void cursor_on(void);
     /*clear screen */
void clear_screen(int, int);
     /* grid */
void grid(int, int, int, int, int, int, int);
char *save_screen_area(int, int, int, int);
void restore_screen_area( char * );
```

2a. The cursor remains off even when you exit the program.

2b. The cursor remains on when the box is displayed. It should look out of place.

2c. The result of commenting both lines out is equivalent to never turning it off. The cursor remains on when the box is displayed.

3. Following is just one of many possible answers:

```
/* Program:  EXER1103.c
 * Author:   Bradley L. Jones
 *           Gregory L. Guntle
 * Purpose:  Use the screen saving/restoring functions.
 *======================================================*/

#include <stdio.h>
#include “tyac.h”

int main()
{
    char c;
    char *screen_buffer;

    grid(4,10,3,20,RED,GREEN,1);
    write_string(“This is a test”, LIGHTBLUE, GREEN, 10,20);
    write_string(“<=  Saving this square area.”,
                   WHITE, BLACK, 7, 40);
    screen_buffer = save_screen_area(4,10,3,20);
    write_string(“Area has been saved”, WHITE, BLACK, 15, 1);
    write_string(“Press any key to continue”,
                   WHITE, BLACK,16,1);
    c = getch();
    clrscr();
    grid(10,20,20,40,WHITE, BLUE,2);
    grid(15,20,15,25,BRIGHTWHITE,CYAN,3);
    printf(“\nPress any key to restore screen\n”);
    c = getch();
    restore_screen_area(screen_buffer);
    return 0;
}
```

4. If you keep calling `save_screen_area()` without freeing the screen, then you will eventually run out of allocatable memory. At this point the program will either end, or your computer will lock up.

Warning: The following program may lock up your computer. This is to illustrate a point. If it does, just turn your computer off and then back on.

```
/* Program:  EXER1104.c
 * Author:   Bradley L. Jones
 *           Gregory L. Guntle
 * Purpose:  Use the screen save function without restoring.
 *
 * NOTE:     THIS IS A BAAAAD Program.  It may lock up your
 *           computer!  It illustrates what can happen if
 *           you don’t free or restore your saved screens.
 *=========================================================*/

#include <stdio.h>
#include “tyac.h”

void save_screen( void );

int main()
{
    char msg[50];
    long ctr = 1;

    clear_screen( YELLOW, BLUE );

    while ( ctr < 1000000 )
    {
      sprintf( msg, “Saving counter: %ld”, ctr );

      grid( 11, 13, 28, 54, BLACK, RED, 2);
      box( 11, 13, 28, 54, DOUBLE_BOX, FILL_BOX, YELLOW, RED );

      write_string(msg, YELLOW, RED, 12, 30);
      ctr++;
    }

    return 0;
}
```

```
void save_screen(void)
{
    char *screen_buffer;

    screen_buffer = save_screen_area( 0, 24, 0, 79 );
    /* nothing done with buffer.... */
    return;
}
```

5. This exercise is on your own; however, similar functions will be presented on later days.

Day 12 Answers

Following are answers to the quiz and exercises.

Quiz Answers

1. A methodology is a set of procedures that are followed to accomplish a task.
2. Scope is the business rules and objectives that constrain a system. The scope of a project is what limits its size.
3. A prototype is a mock-up of what a screen or report will look like.
4. Yes! Even if you make up your own, the idea is to thoroughly think through a system before you start coding it.
5. You should always follow a methodology. The only time to not follow one is when a program is so small and so straightforward that it doesn’t require one. Even then, you generally will think through all the steps.

Exercises Answers

There are no exercise answers for today.

Day 13 Answers

Following are answers to the quiz and exercises.

Quiz Answers

1. Today’s chapter lists several different advantages. These include:
    - ☐ Lower learning curve for people using the application.
    - ☐ Higher comfort level for users in regard to the application.
    - ☐ Consistency among your applications.
    - ☐ Reduced cost in developing.
    - ☐ Increased productivity for you.
2. The F1 function key should provide help information.
3. The F3 function key should exit the program.
4. The F5 function key is available for your use.
5. A well-designed application will only use this combination of colors (yellow on red) when an error has occurred.
6. False, a beep should be used consistently throughout your application. Generally, a beep is used when the user’s attention is needed such as when there is an error.
7. An easy approach to creating your own entry and edit screen is to start with an existing screen and then modify it to your purposes.

Exercises Answers

1. The following is one possible solution. This solution requires that you also include the STDLIB.H header file. This code replaced the case 1 that was in the group file.

```
case 1 :
   strncpy( tmp, groups.date_formed.month, 2 );

   while( valid == FALSE )
   {
     rv = getline( GET_NUM, 0,  6, 17, 0,  2, tmp );

     if( rv == ESC_KEY || rv == F3 || rv == F1 )
     {
```

```
    valid = TRUE;  /* exit loop */
  }
  else
  {
     tmp_nbr = atoi(tmp);
     if( tmp_nbr < 0 || tmp_nbr > 12 )
     {
       /* bad month */
       display_msg_box(“Month must be from 1 to 12”,
                       ct.err_fcol, ct.err_bcol );
     }
     else
     {
       valid = TRUE;
     }
  }
  zero_fill_field(tmp, 2);
  write_string(tmp, ct.fld_fcol, ct.fld_bcol, 6, 17);
}

strncpy( groups.date_formed.month, tmp, 2);
break;
```

2. This exercise is on your own.
3. This exercise is on your own. You should do this exercise because it will be referenced in future days’ exercises.

Day 14 Answers

Following are answers to the quiz and exercises.

Quiz Answers

1. Mnemonic characters are used in menus to allow for quick selection of a specific menu item.
2. The color table will be used in a lot of different programs and functions.

3. The Home key moves the highlight to the first menu option.
4. Unless set up as an exit key, the Page Up and Page Down keys have no effect in a menu.
5. If no box and no shadow is placed on a menu, a menu can be as many lines long as there are rows on the screen.
6. Possible mnemonic values would be 5 and E. You may want to consider X instead of E. Many programs use X for exit.

Exercises Answers

Only Exercise 3 has an answer. One possible answer is as follows:

```
/* Program:  EXER1403.c
 * Author:   Bradley L. Jones
 *           Gregory L. Guntle
 *=========================================================*/

#include <stdio.h>
#include “tyac.h”
#include “colortbl.h”

char *main_menu[10] = {
          “ 1. RED    “, “1Rr”,
          “ 2. YELLOW “, “2Yy”,
          “ 3. BLUE   “, “3Bb”,
          “ 4. GREEN  “, “4Gg”,
          “ 5. Exit   “, “5Ee”  };

char MENU_EXIT_KEYS[MAX_KEYS] = {F3, ESC_KEY};

struct color_table ct;

void initialize_color_table(void);

int main()
{
   int rv;
   int menu_sel=0;

   initialize_color_table();
   clear_screen( ct.bg_fcol, ct.bg_bcol );

   rv = display_menu(10, 30, DOUBLE_BOX, main_menu, 10,
                     MENU_EXIT_KEYS, &menu_sel, NO_LR_ARROW,
                     SHADOW);

   cursor_on();
   return(0);
}

/*------------------------------------------------------*
 * initialize_color_table()                             *
 *                                                      *
 * Set up global color table for rest of application    *
 *------------------------------------------------------*/

void initialize_color_table( void )
{
   ct.bg_fcol = YELLOW;
   ct.bg_bcol = BLUE;

   ct.fld_prmpt_fcol = CYAN;
   ct.fld_prmpt_bcol = BLUE;

   ct.fld_fcol = BRIGHTWHITE;
   ct.fld_bcol = BLUE;

   ct.fld_high_fcol = YELLOW;
   ct.fld_high_bcol = BLUE;

   ct.ttl_fcol = BRIGHTWHITE;
   ct.ttl_bcol = BLACK;

   ct.abar_fcol = BLACK;
   ct.abar_bcol = WHITE;

   ct.menu_fcol = BLACK;
   ct.menu_bcol = WHITE;

   ct.menu_high_fcol = BLUE;
   ct.menu_high_bcol = CYAN;

   ct.err_fcol = YELLOW;
   ct.err_bcol = RED;

   ct.db_fcol = WHITE;
   ct.db_bcol = BROWN;

   ct.help_fcol = YELLOW;
   ct.help_bcol = GREEN;

   ct.shdw_fcol = BLACK;
}
```

Day 15 Answers

Following are answers to the quiz and exercises.

Quiz Answers

1. Action bars are menus containing options for various actions that your program can perform.
2. File, Edit, View, Options, and Help.
3. The File option should be used for actions that apply to an entire file.
4. The Edit option contains actions that modify the current information that is being presented.
5. The View action bar item contains selections that enable you to look in different ways at information from the current file.
6. The Options action bar item is used to present selection items that customize the user interface.
7. The Help option contains actions that are informational.
8. Yes, every action bar should contain Help.
9. The left and right arrows should move from one menu to the next, going either left or right depending on which arrow key is used. If the edge of the action bar is reached, then the arrow should scroll to the other end.
10. The options on an action bar are presented in menus; however, each option provides for a different action that can occur. The name comes from the fact that the options are for different actions.

Exercises Answers

1. Exercises 1 and 3 are for you to do on your own.
2. Following is a listing for the Group Information screens action bar. This listing, GRPSABAR.C, provides the same functionality as Listing 15.3. You'll also need to make the appropriate changes to GROUPS.C to call the action bar. These changes, listed in today's material, included adding the action bar when you draw the screen, setting up F10 as an exit key in `getline()`, and creating an F10 case for when F10 is pressed.

Following is GRPSABAR.C:

```
/*==========================================================
 * Filename: grpsabar.c
 *           RECORD OF RECORDS - Version 1.0
 *
 * Author:   Bradley L. Jones
 *
 * Purpose:  Action bar for groups screen.  This will
 *           contain multiple menus. The functions called
 *           by the menu selections may not be available
 *           until later days.
 *
 * Return:   Return value is either key used to exit a menu
 *           with the exception of F10 which is returned
 *           as Enter to re-display main menu.  In each menu
 *           the left and right keys will exit and move
 *           control to the right or left menu.
 *=========================================================*/


#include <stdio.h>

#include “tyac.h”
#include “records.h”


/*--------------------*
 *     prototypes     *
 *--------------------*/

#include “recofrec.h”

int do_groups_menu1( void );
int do_groups_menu2( void );
int do_groups_menu3( void );
int do_groups_menu4( void );

static void do_something( char * );

/*---------------------------*
 *  groups screen action bar  *
 *---------------------------*/

int do_groups_actionbar( void )
{
  int  menu = 1;
  int  cont = TRUE;
  int  rv   = 0;
  char *abar_text ={“ File   Edit   Search   Help “};

```

```

    while( cont == TRUE )
    {
       write_string(abar_text, ct.abar_fcol, ct.abar_bcol, 1, 2);

       switch( menu )
       {

          case 1:  /* file menu */
                   write_string( “ File “, ct.menu_high_fcol,
                        ct.menu_high_bcol, 1, 2);

                   rv = do_groups_menu1();
                   break;

          case 2:  /* edit menu */
                   write_string( “ Edit “, ct.menu_high_fcol,
                        ct.menu_high_bcol, 1, 9);

                   rv = do_groups_menu2();
                   break;

          case 3:  /* search menu */
                   write_string( “ Search “, ct.menu_high_fcol,
                        ct.menu_high_bcol, 1, 16);

                   rv = do_groups_menu3();
                   break;

          case 4:  /* Help menu */
                   write_string( “ Help “, ct.menu_high_fcol,
                        ct.menu_high_bcol, 1, 25);

                   rv = do_groups_menu4();
                   break;

          default: /* error */
                   cont = FALSE;
                   break;
       }

       switch( rv )
       {
          case LT_ARROW:  menu—;
                          if( menu < 1 )
                              menu = 4;
                          break;

          case RT_ARROW:  menu++;
                          if( menu > 4 )
                              menu = 1;
                        break;

        default:        cont = FALSE;
                        break;
     }
  }
  write_string(abar_text, ct.abar_fcol, ct.abar_bcol, 1, 2);
  cursor_on();
  return(rv);
}



/*--------------------*
 *  do_menu 1  (File) *
 *--------------------*/

int do_groups_menu1( void )
{
  int   rv       = 0;
  int   menu_sel = 0;
  char *saved_screen = NULL;

  char *file_menu[2] = { “ Exit  <F3> “, “1Ee” };
  char exit_keys[MAX_KEYS] = {F3, F10, ESC_KEY};

  saved_screen = save_screen_area( 0, 10, 0, 40 );

  rv = display_menu( 3, 4, SINGLE_BOX, file_menu, 2,
                    exit_keys, &menu_sel, LR_ARROW, SHADOW);

  switch( rv )
  {
     case ENTER_KEY: /* accept selection */
     case CR:
                     rv = F3;
                     break;

     case F3:       /* exiting */
     case ESC_KEY:
     case LT_ARROW: /* arrow keys */
     case RT_ARROW:
                     break;

     case F10:      /* exit action bar */
                     rv = ENTER_KEY;
                     break;

     default:        boop();
                     break;
  }
  restore_screen_area( saved_screen );

  return(rv);
}

/*--------------------*
 *  do_menu 2  (Edit) *
 *--------------------*/

int do_groups_menu2( void )
{
  int   rv       = 0;
  int   menu_sel = 0;
  char *saved_screen = NULL;

  char *edit_menu[8] = {
          “ New            “, “1Nn”,
          “ Add     <F4> “, “2Aa”,
          “ Change  <F5> “, “3Cc”,
          “ Delete  <F6> “, “4Dd” };

  char exit_keys[MAX_KEYS] = {F3, F10, ESC_KEY};

  saved_screen = save_screen_area( 1, 10, 8, 40 );

  rv = display_menu( 3, 11, SINGLE_BOX, edit_menu, 8,
                     exit_keys, &menu_sel, LR_ARROW, SHADOW);

  switch( rv )
  {
     case ENTER_KEY: /* accept selection */
     case CR:
                     switch( menu_sel )
                     {
                       case 1:  /* Clear the screen */
                                do_something( “CLEARING...”);
                                break;


                       case 2:  /* Add a record */
                                do_something( “Adding...”);

                                break;

                       case 3:  /* Update the current record */
                                do_something( “Updating...”);

                                break;

                       case 4:  /* Deleting the current record */
                                do_something( “Deleting...”);

                                break;

                       default: /* continue looping */
                                boop();
                                break;
                     }

                     rv = ENTER_KEY;
                     break;

      case F3:       /* exiting */
      case ESC_KEY:
      case LT_ARROW: /* arrow keys */
      case RT_ARROW:
                     break;

      case F10:      /* action bar */
                     rv = ENTER_KEY;
                     break;

      default:       boop();
                     break;
  }
  restore_screen_area( saved_screen );

  return(rv);
}

/*----------------------*
 *  do menu 3  (Search) *
 *----------------------*/

int do_groups_menu3( void )
{
  int   rv       = 0;
  int   menu_sel = 0;
  char *saved_screen = NULL;

  char *search_menu[6] = {
          “ Find...        “, “1Ff”,
          “ Next      <F7> “, “2Nn”,
          “ Previous  <F8> “, “3Pp” };

  char exit_keys[MAX_KEYS] = {F3, F10, ESC_KEY};

  saved_screen = save_screen_area( 1, 10, 0, 60 );

  rv = display_menu( 3, 18, SINGLE_BOX, search_menu, 6,
                     exit_keys, &menu_sel, LR_ARROW, SHADOW);

  switch( rv )
  {
     case ENTER_KEY: /* accept selection */
     case CR:
                     switch( menu_sel )
                     {
                       case 1:  /* Do find dialog */
                                do_something( “Find...”);

                                break;

                       case 2:  /* Next Record */
                                do_something( “Next...”);

                                break;

                       case 3:  /* Previous Record */
                                do_something( “Previous...”);

                                break;

                       default: /* shouldn’t happen */
                                boop();
                                break;
                     }

                     rv = ENTER_KEY;
                     break;

      case F3:       /* exiting */
      case ESC_KEY:
      case LT_ARROW: /* arrow keys */
      case RT_ARROW:
                     break;

      case F10:      /* action bar */
                     rv = ENTER_KEY;
                     break;

      default:       boop();
                     break;
  }
  restore_screen_area( saved_screen );

  return(rv);
}

/*--------------------*
 *  do menu 4  (Help) *
 *--------------------*/

```

```
int do_groups_menu4( void )
{
  int   rv       = 0;
  int   menu_sel = 0;
  char *saved_screen = NULL;

  char *help_menu[4] = {
          “ Help  <F2> “, “1Hh”,
          “ About      “, “2Ee” };

  char exit_keys[MAX_KEYS] = {F3, F10, ESC_KEY};

  saved_screen = save_screen_area( 1, 10, 0, 60 );

  rv = display_menu( 3, 27, SINGLE_BOX, help_menu, 4,
                     exit_keys, &menu_sel, LR_ARROW, SHADOW);

  switch( rv )
  {
     case ENTER_KEY: /* accept selection */
     case CR:
                     switch( menu_sel )
                     {
                       case 1:  /* Extended Help */
                                display_groups_help();

                                break;

                       case 2:  /* About box */
                                do_something( “About box...”);

                                break;

                       default: /* continue looping */
                                boop();
                                break;
                     }

                     break;

      case F3:       /* exiting */
      case ESC_KEY:
      case LT_ARROW: /* arrow keys */
      case RT_ARROW:
                     break;

      case F10:      /* action bar */
                     rv = ENTER_KEY;
                     break;

      default:       boop();
```

```
                     break;
  }
  restore_screen_area( saved_screen );

  return(rv);
}


/*------------------------------------------------------*
 *   Generic function - temporary                       *
 *-----------------------------------------------------*/
static void do_something(char *msg)
{
  display_msg_box( msg, ct.help_fcol, ct.help_bcol );
}
```

Day 16 Answers

Following are answers to the quiz and exercises.

Quiz Answers

1. If an application requires no explanation, then help isn't necessary. It's doubtful that an application is simple enough that help is not needed.
2. There are several types of help:
    - ☐ Startup help
    - ☐ General help
    - ☐ Help for help
    - ☐ Extended help
    - ☐ Help for keys
    - ☐ Help index
    - ☐ About... help
    - ☐ Context-sensitive help
3. No, tutorials aren't considered help. They provide the user help in using an application.
4. Startup help should be used by applications that require command line parameters or applications that don't have any screens to present to the user.

5. If only extended help is being provided, then F1 should be used.
6. Context-sensitive help should always be assigned to F1. This means that extended help should be assigned to F2.

Exercises Answers

All of today’s exercises are on your own. You should refer to today’s listings.

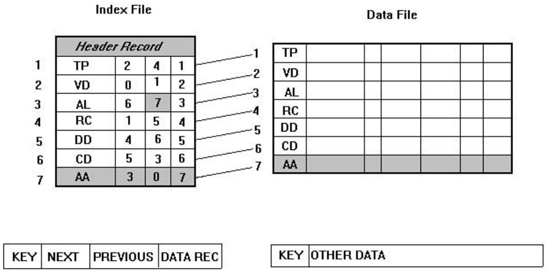
Day 17 Answers

Following are answers to the quiz and exercises.

Quiz Answers

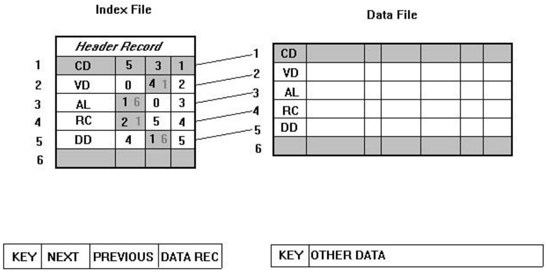
1. Many programmers use the term I/O to stand for any action that results in information flowing into or out of a computer program. Input is information flowing in; output is information flowing out.
2. A field is an individual piece of data.
3. A record is a group of related fields.
4. A data file is a group of related records that are stored on a secondary medium such as a disk drive.
5. A database is a group of related data files.
6. An index file is a separate file from the data file. Because an index contains a smaller amount of information, more of it can be read and manipulated quicker than the data file itself. Using an index allows quick access to a large data file.
7. If you are using text mode, then several translations will occur automatically as you work with the data. A file opened in binary mode doesn’t do the translations of a text mode file.
8. If a preexisting file is opened with “w+b”, then it is being opened for reading and writing in binary mode. Because the “w+” was used, a preexisting file will be cleared. If “r+” had been used instead, then preexisting data would have been retained.

9.



	Header Record			
1	TP	2	4	1
2	VD	0	1	2
3	AL	6	7	3
4	RC	1	5	4
5	DD	4	6	5
6	CD	5	3	6
7	AA	3	0	7

10.



	Header Record			
1	CD	5	3	1
2	VD	0	4 1	2
3	AL	1 6	0	3
4	RC	2 1	5	4
5	DD	4	1 6	5
6				

Exercises Answers

1. There isn’t an answer to this exercise.
2. The Group files index should look like the following:

```
typedef struct
{
   char group[25+1];
   unsigned short prev;
   unsigned short next;
   unsigned short data;

} GROUP_INDEX;
```

3. The disk will contain the completed code for the groups entry and edit screen.

Day 18 Answers

Following are answers to the quiz and exercises.

Quiz Answers

1. `clear` and `find...`
2. The `clear` function clears the screen so that a new record can be entered. The `find...` function finds a corresponding record for the data in the key field on the screen.
3. You should always ask the users if they are sure they want to delete the record.
4. The delete flag is used to mark a record as deleted. This is used in place of physically deleting a record. It gives you the ability to include an undelete function.
5. On the medium code screen, the page up and page down functions place the cursor on the first or last field on the screen.
6. On the musical items screen, the page up and page down keys display the next or previous set of songs for the given musical item.
7. The code is shown with the capability to hold 13 pages of seven songs for a total of 91 songs. You could modify this to display as many songs as memory would permit.

Exercises Answers

1. There is not an answer to this exercise.
2. The following is one possible answer. This program needs to be linked with the TYAC.LIB library.

```
/*==========================================================
 * Filename: DUMPIDX.c
 *
 * Author:   Bradley L. Jones
 *           Gregory L. Guntle
 *
 * Purpose:  Dump the index information for the album files
 *=========================================================*/

#include <stdio.h>
```

```
#include <conio.h>         /* not an ANSI header, used for getch()
*/
#include <string.h>        /* for strlen() */
#include <ctype.h>
#include “tyac.h”
#include “records.h”

/*--------------------*
 *     prototypes     *
 *--------------------*/
#include “recofrec.h”

/*--------------------*
 *  Global Variables  *
 *--------------------*/
FILE *idx_fp;            /* Index File Ptr */
FILE *db_fp;             /* Data file */
FILE *song_fp;           /* For tracking songs */
int  nbr_records;         /* Total # of recs */
int  start_rec;           /* Starting rec for alpha */
int  disp_rec;
int  total_songs;         /* Nbr of songs in the song file */

ALBUM_REC albums;
ALBUM_REC *p_albums = &albums;
ALBUM_INDEX alb_idx;

/*--------------------------*
 * define global variables  *
 *--------------------------*/

#define ONE_SONG    sizeof(songs[0])
#define SEVEN_SONGS sizeof(songs)

#define ALBUMS_DBF   “ALBUMS.DBF”
#define ALBUMS_IDX   “ALBUMS.IDX”

/*=======================================================*
 *                     main()                            *
 *=======================================================*/

int main(void)
{
  int rv       = 0;
  int srch_rec = 0;

  /* Open both Index and DBF file */
  if ( (rv = open_files(ALBUMS_IDX, ALBUMS_DBF)) == 0 )
  {
    printf(“Nbr records = %2d   Start_rec = %2d\n\n”,
       nbr_records, start_rec);
    printf(“ rv  sr  df  ns   p   n   d   s\n”);
    printf(“ --  --  --  --  --  --  --  --\n”);
    srch_rec = start_rec;
    while (srch_rec !=0)
    {
      rv = get_rec(srch_rec, idx_fp, sizeof(alb_idx),
                sizeof(int)*2, (char *)&alb_idx);
      printf(“ %2d  %2d  %2d  %2d  %2d  %2d  %2d  %2d  %s\n”,
             rv, srch_rec, alb_idx.del_flag, alb_idx.nbr_songs,
             alb_idx.prev, alb_idx.next, alb_idx.data,
             alb_idx.song, alb_idx.title);
      srch_rec = alb_idx.next;
    }
    rv = close_files();      /* Close IDX and DBF file */
  }

  return 0;
}

/*==========================================================*
 *                    end of listing                        *
 *==========================================================*/
```

```
Nbr records =  5   Start_rec =  2
```

Output

```
 rv  sr  df  ns   p   n   d   s
 —   —   —   —   —   —   —   —
  0   2   0  21   0   1   2  15  10,000 Maniacs: Unplugged
  0   1   0  14   2   4   1   1  Common Thread
  0   4   0   7   1   5   4  43  Violent Femmes
  0   5   0  14   4   0   5  50  White Wedding
```

Analysis

rv = return value

sr = search record

df = delete flag

ns = number of songs

p = previous song record

n = next song record

d = data file record

s = first song record

3. This exercise is on your own.

Day 19 Answers

Following are answers to the quiz and exercises.

Quiz Answers

1. By prototyping a report, you obtain an idea of what the report will look like. It is easier to change a prototype than it is to change code.
2. A list is a report that lists data from a file.
3. Lists are good for verifying that each database contains the information that you expect.
4. `9`s are used on prototypes as placeholders for numbers. Where a `9` appears on a prototype, a number field will be.
5. `X`s are used on prototypes as placeholders for alphanumeric characters. Where an `X` appears, an alphanumeric field will be.
6. So that you know when the report was printed. Data could have changed.
7. "Real" data on a prototype presents a view of what the report is really going to look like. Spacing appears different when `X`s and `9`s are used.
8. Many printers can write up to 60 lines on a page. Many people stop at 55 just to be safe.
9. The following would be printed for the names:

```
Foster , MaryB E.
Bell   , Mike J.
LiVenst, Angela  .
```

You would want to include instructions with the prototype (in a specification) that spaces should be suppressed. Additionally, the period following the middle initial should only be printed if a middle name exists.

Exercises Answers

1.
```
                List of Albums
                   99/99/99

Title                              Group
------------------------------     -------------------------
XXXXXXXXXXXXXXXXXXXXXXXXXXXXXX     XXXXXXXXXXXXXXXXXXXXXXXXX
XXXXXXXXXXXXXXXXXXXXXXXXXXXXXX     XXXXXXXXXXXXXXXXXXXXXXXXX
XXXXXXXXXXXXXXXXXXXXXXXXXXXXXX     XXXXXXXXXXXXXXXXXXXXXXXXX
Songs From the Big Chair           Tears for Fears
Back in Black                      AC/DC
```

2. The following is the List of Musical Items Report based on the specification in Exercise 1.

```
/*========================================================
 * Filename: listalb.c
 *
 * Author:   Bradley L. Jones
 *           Gregory L. Guntle
 *
 * Purpose:  Lists the information in the albums DBF.
 *
 *======================================================*/

#include <stdio.h>
#include <string.h>
#include “tyac.h”
#include “records.h”

/*--------------------*
 *  Global Variables  *
 *--------------------*/
extern FILE *idx_fp;      /* Main File ptr to data file */
extern FILE *db_fp;       /* Data file */
extern nbr_records;       /* Total # of rec for mediums */
extern int start_rec;
extern int disp_rec;

static int line_ctr;             /* Line ctr */
static int page_ctr;             /* Page ctr */

/*------------------*
 * Defined constants*
 *------------------*/
#define ALBUMS_IDX   “ALBUMS.IDX”
#define ALBUMS_DBF   “ALBUMS.DBF”

/*------------------------*
 * structure declarations *
 *------------------------*/
extern ALBUM_REC albums;

/*--------------------*
 *  Prototypes        *
 *--------------------*/
#include “recofrec.h”

void print_alb_hdr(void);
int  process_albums(void);
void print_album(void);

void setup_date(void);
extern char today[8];

/*======================================================*
 *   list_albums()                                      *
 *======================================================*/

int list_musical_items(void)
{
  int rv=NO_ERROR;

  /* Open both Index and DBF file */
  if ( (rv = open_files(ALBUMS_IDX, ALBUMS_DBF)) == 0 )
  {
     /* Are there any records to process ? */
     if (nbr_records == 0)
     {
       display_msg_box(“No album records to process”,
          ct.err_fcol, ct.err_bcol);
     }
     else
     {
       setup_today();
       print_alb_hdr();
       rv = process_albums();
     }               /* End ELSE - records to process */

     rv = close_files();
  }                  /* End No Errors on Opening Files */
  return(rv);
}

/*=======================================================*
 *   print_alb_hdr()                                     *
 *=======================================================*/
void print_alb_hdr()
{
  fprintf(stdprn,”\n\r\t\tList of Albums\n\r”);
  fprintf(stdprn,”\t\t   %s\n\r\n\r”, today);

  fprintf(stdprn,”Title Group\n”);
  fprintf(stdprn,”---------------------------”);
  fprintf(stdprn,”     ------------------------\n\r”);
}

/*=======================================================*
 *   process_albums()                                    *
 *=======================================================*/
int process_albums()
{
  int rv = NO_ERROR;
  static int done = FALSE;
  static int srch_rec;
  static ALBUM_INDEX temp;

  line_ctr = 0;
  page_ctr = 1;

  srch_rec = start_rec;
  while (rv == NO_ERROR && !done)
  {
    /* Get INDEX */
    rv = get_rec(srch_rec, idx_fp, sizeof(temp),
         sizeof(int)*2, (char *)&temp);
    if (rv == NO_ERROR)
    {
      /* Get the data record */
      rv = get_rec(temp.data, db_fp, sizeof(albums),
              0, (char *)&albums);
      if (rv == NO_ERROR)
      {
        print_album();
        srch_rec = temp.next;
        /* Check for end of list */
        if (srch_rec == 0)
        {
           done = TRUE;
           fprintf(stdprn, “\f”);
        }
        else
           done = FALSE;
      }         /* End of NO_ERROR - from DBF */
    }           /* End of NO_ERROR - from INDEX */
  }             /* End WHILE */
  return(rv);
}

/*======================================================*
 *   print_album()                                      *
 *======================================================*/
void print_album()
{
  int i;
  char hold_date[8];
```

```

   if (line_ctr+1 > 55)
   {
      fprintf(stdprn,”\f”);     /* New page */
      print_alb_hdr();          /* Reprint header */
      line_ctr = 6;             /* for heading space */
      page_ctr++;
   }
   else
   {
      line_ctr++;
   }

   fprintf(stdprn,”%-30s     %-25s\n\r”,
                     albums.title, albums.group );
}
```

3. Exercise 3 is on your own.
4. Exercise 4 is on your own.

Day 20 Answers

Following are answers to the quiz and exercises.

Quiz Answers

1. Free-form testing does not have any written objectives or requirements. The objective is to “bang on the software and try to break it.”
2. The four categories of testing are:
    a. unit testing
    b. integration testing
    c. alpha testing
    d. beta testing
3. In the same order mentioned in Answer 2.
4. When you are confident that the program is as accurate as it can be.

5. A mistake or error in a computer program.
6. Syntax errors are errors in the syntax or code that cause the compiler to have problems.
7. Syntax errors, logic errors, and recursive errors. Recursive errors are actually syntax and logic errors.
8. Removing bugs (errors or problems) from code.
9. A patch is a fix that is sent out because of an error in a program. A patch usually will not be charged for, nor should it cause the version number to change. An upgrade is a release of the software that incorporates patches, updates, and enhancements to the software. It is released with a new version number. In addition, an upgrade may have a fee associated with it.
10. A software package that is released to the public may never be complete. People will always have suggestions for updates and upgrades to the software.

Exercises Answers

All of today's exercises are on your own.

Day 21 Answers

Following are answers to the quiz and exercises.

Quiz Answers

1. No! There is always more to learn in any language.
2. There are several areas in which you can increase your C knowledge. These include graphics programming and advanced applications development. You can also increase your skills in more technical areas such as compiler development and network programming.
3. C++ and assembler.
4. There are several benefits to using assembler. Among them are speed, additional control, and the capability to write more compact programs.
5. An object-oriented language is identified by its capability to encapsulate, polymorph, and inherit.

6. No. C is a subset of C++. Not all C++ constructs are usable in C.
7. Yes. C++ is a superset of C. Everything that can be used in C can be used in C++; however, not everything that can be used in C should be used in C++.
8. The most commonly used platforms are DOS, UNIX, Windows, OS/2, and System 7 (Macintosh). There are C compilers for all of these.

Exercises Answers

All of today's exercises are on your own.