ARRAYS

An array is composed of a series of elements of the same data type. Each element is contiguous in memory. That is, each element in the array is adjacent to another element of the same array. Arrays are useful for storing data of the same data type under one unique name. For example, an array of student grades, each of type `int`, might be stored in an integer array called `grades[]`. An array of characters in a person's first name, each of type char, might be stored in a character array called `firstName[]`.

An array is declared by specifying its type, name, and dimension. So, the two arrays mentioned above might be declared as follows:

```
int grades[50];
char firstName[25];
```

In the example above, `grades[]` is declared as an integer array, and it is capable of holding grades for 50 students. `firstName[]`is declared as a character array, and is capable of holding 25 characters for the first name. Often, a symbolic constant will be used to dimension an array rather than a literal number. For example, the arrays are better declared as follows:

```
#define TOTAL_STUDENTS   50
#define FIRST_NAME_MAX   25

int grades[TOTAL_STUDENTS];
char firstName[FIRST_NAME_MAX];
```

Again, the memory is stored in a contiguous fashion. This means that we can access any value in the array simply by a process of indexing. Arrays in C Programming always begin with in index of 0. The process of indexing allows us to use arrays with looping statements to make coding simple and easy to read. For example, we can enter in grades for all 50 students with a few lines of code:

```
for(counter = 0; counter < TOTAL_STUDENTS; counter++)
{
     printf("\nEnter grade #%02d: ", counter + 1);
     scanf("%d", &grades[counter]);
}
```

And to print out all grades:

```
for(counter = 0; counter < TOTAL_STUDENTS; counter++)
     printf("Grade #%02d = %d\n",
          counter + 1, grades[counter]);
```

Arrays In Memory

Arrays are stored in contiguous memory. That is, memory equal to the size of the first element data type is reserved for the first array element, memory equal to the size of the second element data type is reserved for the second array element, and so on. In the case of a character array, the size of each element data type is one byte. In the case of an integer array, the size of each element data type is four bytes. The memory for the `grades[]` and `firstName[]` arrays might appear in memory as shown below:

ADDRESS	CONTENTS	NAME
0x00000000	…	…
0x00000001	...	...
0x00000002	...	...
0x00000003	...	...
…	...	...
0x00A02F00	‘J’	firstName[0]
0x00A02F01	‘o’	firstName[1]
0x00A02F02	‘e’	firstName[2]
0x00A02F03	‘\0’	firstName[3]
0x00A02F04	‘k’	firstName[4]
0x00A02F05	‘\0’	firstName[5]
...	...	...
0x00A02F18	‘s’	firstName[24]
0x00A02F19	…	…
0x00A02F1A	...	...
...	...	...
0x00B0005E	...	...
0x00B0005F	...	...
0x00B00060	91	grades[0]
0x00B00061		
0x00B00062		
0x00B00063		
0x00B00064	83	grades[1]
0x00B00065		
0x00B00066		
0x00B00067		
…	…	…
0x00B00124	100	grades[49]
0x00B00125		
0x00B00126		
0x00B00127		
0x00B00128	....	...
0x00B00129	...	...

```
#include <stdio.h>

#define NUM_HOMEWORKS   6
#define NUM_HOUR_EXAMS  2

#define WT_HOMEWORK     ((100. / 40.) * 0.30)
#define WT_HOUR_EXAMS   0.25
#define WT_PROJECT      0.20
#define WT_FINAL_EXAM   0.25

/*-------------------------------------------------------------------
 * Function Name: main()
 *
 * Description:   This program will request student grade data and
 *                calculate the course average for the student.
 *                The grade material (homeworks, hour exams,
 *                project, final exam) are each weighted differently
 *                when calculating the average for the course.
 *
 * Inputs:        None
 *
 * Output:        None
 *
 * Return:        Status
 *-------------------------------------------------------------------*/
int main(void)
{
      double courseAvg, homeworkAvg, hourExamAvg;
      int homework[NUM_HOMEWORKS];
      int hourExam[NUM_HOUR_EXAMS];
      int project;
      int finalExam;
      int counter;
      char userInput;

      do
      {
            /* Get the student's homework grades */
            for(counter = 0; counter < NUM_HOMEWORKS; counter++)
            {
                  printf("Enter Homework #%d Grade: ", counter + 1);
                  scanf("%d", &homework[counter]);
            }

            /* Get the student's hour exam grades */
            for(counter = 0; counter < NUM_HOUR_EXAMS; counter++)
            {
                  printf("Enter Hour Exam #%d Grade: ", counter + 1);
                  scanf("%d", &hourExam[counter]);
            }

            /* Get the student's project grade */
            printf("Enter Project Grade: ");
            scanf("%d", &project);
```

```
            /* Get the student's final exam grade */
            printf("Enter Final Exam Grade: ");
            scanf("%d", &finalExam);

            /* Calculate the homework average */
            for(counter=0, homeworkAvg=0; counter < NUM_HOMEWORKS; counter++)
                  homeworkAvg += homework[counter];
            homeworkAvg /= NUM_HOMEWORKS;

            /* Calculate the hour exam average */
            for(counter=0, hourExamAvg=0; counter < NUM_HOUR_EXAMS; counter++)
                  hourExamAvg += hourExam[counter];
            hourExamAvg /= NUM_HOUR_EXAMS;

            /* Calculate the course grade average */
            courseAvg = (homeworkAvg * WT_HOMEWORK)   + \
                        (hourExamAvg * WT_HOUR_EXAMS) + \
                        (project * WT_PROJECT)        + \
                        (finalExam * WT_FINAL_EXAM);

            /* Display the course average grade */
            printf("\nCOURSE AVERAGE: %.1f%%\n", courseAvg);

            printf("\nEnter another student (Y or N)? ");
            scanf(" %c", &userInput);
      } while( (userInput == 'Y') || (userInput == 'y') );

      printf("\nProgram Complete...\n");
      return 0;
}
```

Sample Run,

```
Enter Homework #1 Grade: 40
Enter Homework #2 Grade: 40
Enter Homework #3 Grade: 40
Enter Homework #4 Grade: 40
Enter Homework #5 Grade: 40
Enter Homework #6 Grade: 40
Enter Hour Exam #1 Grade: 100
Enter Hour Exam #2 Grade: 100
Enter Project Grade: 100
Enter Final Exam Grade: 100

COURSE AVERAGE: 100.0%

Enter another student (Y or N)? y
Enter Homework #1 Grade: 0
Enter Homework #2 Grade: 0
Enter Homework #3 Grade: 0
Enter Homework #4 Grade: 0
Enter Homework #5 Grade: 0
Enter Homework #6 Grade: 0
Enter Hour Exam #1 Grade: 0
Enter Hour Exam #2 Grade: 0
Enter Project Grade: 0
Enter Final Exam Grade: 0

COURSE AVERAGE: 0.0%

Enter another student (Y or N)? y
Enter Homework #1 Grade: 20
Enter Homework #2 Grade: 20
Enter Homework #3 Grade: 20
Enter Homework #4 Grade: 20
Enter Homework #5 Grade: 20
Enter Homework #6 Grade: 20
Enter Hour Exam #1 Grade: 50
Enter Hour Exam #2 Grade: 50
Enter Project Grade: 50
Enter Final Exam Grade: 50

COURSE AVERAGE: 50.0%

Enter another student (Y or N)? n

Program Complete...
Press any key to continue
```

Initialization of Arrays
Arrays may be initialized in the declaration if desired by the programmer. Examples of array initialization are shown below:

```
      int daysPerMonth[12] = {31, 28, 31, 30, 31, 30, 31, 31, 30, 31, 30, 31};
OR
      int daysPerMonth[] = {31, 28, 31, 30, 31, 30, 31, 31, 30, 31, 30, 31};


      char firstName[4] = {'J', 'o', 'e', '\0'};
OR
      char firstName[] = {'J', 'o', 'e', '\0'};
OR
      char firstName[4] = "Joe";
OR
      char firstName[] = "Joe";
```

Passing Arrays to Functions
Arrays may be initialized passed to functions just as any other variable. Remember, however, that the argument is copied into the formal parameter of the function. In the case of the array, the argument is the address of the first element of the array, not the value at the location. Thus, any change made to the contents of the array (i.e., the value at a location) results in a permanent change to the array after the function call completes.

This can be useful in our design if it is desirable to have the function perform more than one operation that has lasting effect for the duration of the program. Remember, a function can only return one value, but by passing an array to a function and then changing the array, we can "fake-out" the code by making it appear the function is doing more than just returning one value. We say that the function output is a change in the array that is passed in. Remember, though, that the function is still only capable of returning one value and that is the function return. This technique describes the difference between "function output" and "function return" as described in the function description header.

It might also be desirable to pass an array to a function, but yet have the function perform no changes on the array. For instance, if we are calculating a grade average, we would not want the function to have the ability to change any of the grades during its computation. To safeguard against this behavior, we use the `const` parameter modifier.

Examples of Function Declarations:

```
      void FunctionA(int num, int array[]);
      void FunctionB(int num, const int array[]);
```

Example of Function Definition:

```
      int FunctionB(int num, const int array[])
      {
            printf("Element %d of the array is %d\n", num, array[num]);
      }
```

Multi-Dimensional Arrays
Arrays may be declared multi-dimensional simply by providing the number of dimensions at the time of declaration. Each dimension is defined by `[n]` where `n` is an integer that specifies the number of elements in the dimension.

```
For example,
      #define X_POS     4
      #define Y_POS     3
      #define Z_POS     2

      #define NUM_STUDENTS    2
      #define NAME_LENGTH     80

      int xyCoordinates[X_POS][Y_POS];          /* 2D Array */

      int xzyCoordinates[X_POS][Y_POS][Z_POS];  /* 3D Array */

      char students[NUM_STUDENTS][NAME_LENGTH]; /* 2D Array */
```

Initialization of Multi-Dimensional Arrays
Multi-Dimensional Arrays may be initialized in the declaration if desired by the programmer. The more dimensions of the array, the trickier it is to initialize. If you think like the compiler, which stores the array starting from its right-most dimension, this may help you figure out the correct syntax. In the examples below, pay close attention to the way the multi-dimensional arrays are initialized. Look at the curly brackets in the initialization. They are important and used in the initialization in such a manner that is consistent with the compiler. Improper use of the curly braces will result in an array that is not initialized correctly; that is, you will not get a compiler error, but the array will not be initialized as you planned. Careful attention must be paid when initializing multi-dimensional arrays. Examples of multi-dimensional array initialization are shown below:

```
Examples of Multi-Dimensional Array Initializations:
int xyCoordinates[X_POS][Y_POS] =  { \
                              {0,1,2}, {3,4,5}, {6,7,8}, {9,10,11} };

OR

int xyCoordinates[X_POS][Y_POS] = { \
                              {0,1,2},
                              {3,4,5},
                              {6,7,8},
                              {9,10,11} };

OR

int xyCoordinates[][Y_POS] = { \
                              {0,1,2},
                              {3,4,5},
                              {6,7,8},
                              {9,10,11} };
```

```
int xyzCoordinates[X_POS][Y_POS][Z_POS] = { \
            {{ 0, 1}, { 2, 3}, { 4, 5}},
            {{ 6, 7}, { 8, 9}, {10,11}},
            {{12,13}, {14,15}, {16,17}},
            {{18,19}, {20,21}, {22,23}} };

OR

int xyzCoordinates[][Y_POS][Z_POS] = { \
            {{ 0, 1}, { 2, 3}, { 4, 5}},
            {{ 6, 7}, { 8, 9}, {10,11}},
            {{12,13}, {14,15}, {16,17}},
            {{18,19}, {20,21}, {22,23}} };
```

```
char students[NUM_STUDENTS][NAME_LENGTH] = { \
                        {“John Q. Doe”},
                        {“Jane S. Smith”} };

OR

char students[][NAME_LENGTH] = { \
                        {“John Q. Doe”},
                        {“Jane S. Smith”} };
```

Note: In the example above regarding the `student[]` array, the compiler will still use `NAME_LENGTH` characters for each of the `NUM_STUDENTS` students, as would be expected. However, the names are initialized with the characters shown in each name string. The character immediately following each student name is set to NULL and the remaining characters in the character array have a value that does not matter. The NULL terminator within the character array defines the actual name string within the array. Thus, printing out the string will mean that the characters up to the NULL character will be printed and the remaining characters will be ignored.

A program that displays the initialized arrays follows.

```
#include <stdio.h>

#define X_POS     4
#define Y_POS     3
#define Z_POS     2

#define NUM_STUDENTS    2
#define NAME_LENGTH     80

/*-------------------------------------------------------------------
 * Function Name: main()
 *
 * Description:   This program will demonstrate the initialization and
 *                display of multi-dimensional arrays.
 *
 * Inputs:        None
 *
 * Output:        None
 *
 * Return:        Status
 *-------------------------------------------------------------------*/
int main(void)
{
      int xyCoordinates[][Y_POS] = { \
                              {0,1,2},
                              {3,4,5},
                              {6,7,8},
                              {9,10,11} };

      int xyzCoordinates[][Y_POS][Z_POS] = { \
                              {{0,1}, {2,3}, {4,5}},
                              {{6,7}, {8,9}, {10,11}},
                              {{12,13}, {14,15}, {16,17}},
                              {{18,19}, {20,21},{22,23}} };

      char students[][NAME_LENGTH] = { \
                              {"John Q. Doe"},
                              {"Jane S. Smith"} };

      int x, y, z; /* Counter Variables */

      printf("\nTHE X-Y COORDINATES 2D ARRAY\n");
      for(x = 0; x < X_POS; x++)
            for(y = 0; y < Y_POS; y++)
                  printf("xyCoordinates[%d][%d] = %d\n",
                              x, y, xyCoordinates[x][y]);

      printf("\nTHE X-Y-Z COORDINATES 3D ARRAY\n");
      for(x = 0; x < X_POS; x++)
            for(y = 0; y < Y_POS; y++)
                  for(z = 0; z < Z_POS; z++)
                        printf("xyzCoordinates[%d][%d][%d] = %d\n",
                              x, y, z, xyzCoordinates[x][y][z]);
```

```
      printf("\nTHE STUDENTS 2D ARRAY\n");
      for(x = 0; x < NUM_STUDENTS; x++)
            printf("students[%d] = %s\n", x,students[x]);

      return 0;
}
```

Sample Run,

```
THE X-Y COORDINATES 2D ARRAY
xyCoordinates[0][0] = 0
xyCoordinates[0][1] = 1
xyCoordinates[0][2] = 2
xyCoordinates[1][0] = 3
xyCoordinates[1][1] = 4
xyCoordinates[1][2] = 5
xyCoordinates[2][0] = 6
xyCoordinates[2][1] = 7
xyCoordinates[2][2] = 8
xyCoordinates[3][0] = 9
xyCoordinates[3][1] = 10
xyCoordinates[3][2] = 11

THE X-Y-Z COORDINATES 3D ARRAY
xyzCoordinates[0][0][0] = 0
xyzCoordinates[0][0][1] = 1
xyzCoordinates[0][1][0] = 2
xyzCoordinates[0][1][1] = 3
xyzCoordinates[0][2][0] = 4
xyzCoordinates[0][2][1] = 5
xyzCoordinates[1][0][0] = 6
xyzCoordinates[1][0][1] = 7
xyzCoordinates[1][1][0] = 8
xyzCoordinates[1][1][1] = 9
xyzCoordinates[1][2][0] = 10
xyzCoordinates[1][2][1] = 11
xyzCoordinates[2][0][0] = 12
xyzCoordinates[2][0][1] = 13
xyzCoordinates[2][1][0] = 14
xyzCoordinates[2][1][1] = 15
xyzCoordinates[2][2][0] = 16
xyzCoordinates[2][2][1] = 17
xyzCoordinates[3][0][0] = 18
xyzCoordinates[3][0][1] = 19
xyzCoordinates[3][1][0] = 20
xyzCoordinates[3][1][1] = 21
xyzCoordinates[3][2][0] = 22
xyzCoordinates[3][2][1] = 23

THE STUDENTS 2D ARRAY
students[0] = John Q. Doe
students[1] = Jane S. Smith
Press any key to continue
```

CHARACTER STRINGS

A character string is an array of characters terminated with a NULL character (`'\0'`). The NULL character terminates the character array to mark the end of the string. A typical character string might appear in memory as shown below.

U	M	A	S	S		L	O	W	E	L	L	\0

The character string above can be initialized in its declaration as shown below.

```
char school[] = {'U','M','A','S','S',' ','L','O','W','E','L','L','\0'};
```

If we provide a dimension greater than the amount of characters needed to hold the string plus its NULL, the characters beyond the NULL character are themselves equal to NULL. Thus, the declaration,

```
char school[100] = \
      {'U','M','A','S','S',' ','L','O','W','E','L','L','\0'};
```

will create a character array called school[] and the first characters of the array will be filled with those characters in the declaration. All remaining characters in the array will be equal to NULL.

The preferred way to initialize a string at the declaration is shown below:

```
char school[] = "UMASS LOWELL";
```

Note that the total number of characters is equal to the sum of the characters in the declaration PLUS one additional character position to hold the terminating NULL character of the string. If there is no terminating NULL character, then we do not have a valid C string.

The ANSI C Programming library has many functions that operate on character strings and characters. Some of the more useful functions are presented in the tables that follow.

STRING FUNCTIONS

You must include the string.h header file.
#inlcude <string.h>

STRING FUNCTION	DESCRIPTION	RETURN
gets(s)	Gets a string from the keyboard (up to the '\n' character) and stores the string in s.	Pointer to s or NULL pointer on error
puts(s)	Print the string s followed by a '\n'.	Non-Negative integer or EOF on error
strcpy(s1, s2)	Copies the contents of s2 into s1 and appends a NULL character to s1.	Pointer to s1
strcat(s1, s2)	Concatenates s2 onto the end of s1 and appends a NULL character to s1.	Pointer to s1
strlen(s)	Determines the length of s. Note: the NULL character is not counted as part of the length of the string.	Integer length of the string not including the NULL character.
strchr(s, c)	Locates the first occurrence of c in the string pointed to by s.	Pointer to the location or a NULL pointer if no match is found.
strstr(s1, s2)	Locates the first occurrence of string pointed to by s2 in the string pointed to by s1.	Pointer to the location or a NULL pointer if no match is found.
strtok(s1, s2)	Breaks up s1 into a sequence of tokens, each of which is delimited by a character from the string pointed to by s2. The first call in the sequence has s1 as its first argument, and is followed by calls with a NULL pointer as their first argument. The separator string, s2, may be different from call to call.	Returns a pointer to the first character of a token or a NULL pointer if there are no tokens. For example, if s1 is the string "abcd" and "c" is the separator string, s2, then a pointer to "a" is returned within s1 and the printed token is "ab".
strcmp(s1, s2)	Determines if s1 is equal to s2. The comparison is done in lexicographic order.	Integer value less than 0 if s1 is less than s2, equal to 0 if s1 is equal to s2, and greater than 0 if s1 is greater than s2.

An example using three the less obvious string functions `strchr(), strstr(),` and `strtok()` is provided below.

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

int main(void)
{
      static char dayTime[] = "10-19-2004,16:30:25";
      char *t;

      t=strchr(dayTime, ',');
      printf("t = %s\n", t);

      t=strstr(dayTime, "00");
      printf("t = %s\n", t);

      t=strtok(dayTime,"-,:");
      printf("month = %s\n", t);

      t=strtok(NULL,"-,:");
      printf("day = %s\n", t);

      t=strtok(NULL,"-,:");
      printf("year = %s\n", t);

      t=strtok(NULL,"-,:");
      printf("hours = %s\n", t);

      t=strtok(NULL,"-,:");
      printf("minutes = %s\n", t);

      t=strtok(NULL,"-,:");
      printf("seconds = %s\n", t);

      t=strtok(NULL,"-,:");
      printf("End = %s\n", t);

      return 0;
}
```

Sample Run,

```
Y:\apc-cc-conduit\Images>a
t = ,16:30:25
t = 004,16:30:25
month = 10
day = 19
year = 2004
hours = 16
minutes = 30
seconds = 25
End = (null)
```

```
Y:\apc-cc-conduit\Images>
```

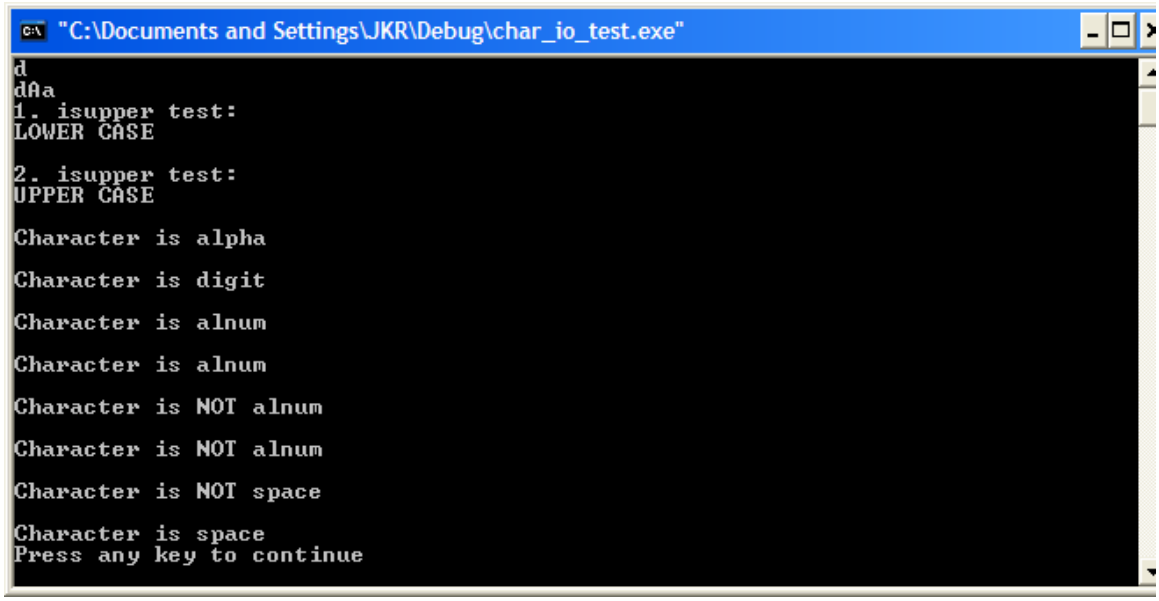
CHARACTER FUNCTIONS

You must include the ctype header file.
#include <ctype.h>

CHARACTER FUNCTIONS	DESCRIPTION
getchar()	Read a char from the keyboard and return the char. Usage Example: c = getchar();
putchar(c)	Print the char c to the monitor.
toupper(c)	Takes a char as an argument and returns the upper-cased version of the argument. The char, c, is unaffected.
tolower(c)	Takes a char as an argument and returns the lower-cased version of the argument. The char, c, is unaffected.
isupper(c)	Takes a char as an argument and returns Boolean true if the char is upper-cased. The char, c, is unaffected.
islower(c)	Takes a char as an argument and returns Boolean true if the char is lower-cased. The char, c, is unaffected.
isalpha(c)	Takes a char as an argument and returns Boolean true if the char is a letter of the alphabet. The char, c, is unaffected.
isdigit(c)	Takes a char as an argument and returns Boolean true if the char is a character between ‘0’ and ‘9’. The char, c, is unaffected.
isalnum(c)	Takes a char as an argument and returns Boolean true if the char is a character that is a letter of the alphabet or is a character that is between ‘0’ and ‘9’. The char, c, is unaffected.
isspace(c)	Takes a char as an argument and returns Boolean true if the char is a whitespace character. The char, c, is unaffected.

```
#include <stdio.h>
#include <ctype.h>

int main(void)
{
      char c1, c2, c3;

      c1 = getchar();

      putchar(c1);

      c3 = 'a';
      c2 = toupper(c3);
      putchar(c2);

      c2 = tolower(c2);
      putchar(c2);

      printf("\n1. isupper test: ");
      if(isupper(c2))
            printf("\nUPPER CASE\n");
      else if(islower(c2))
            printf("\nLOWER CASE\n");
      else
            printf("\nwoops!\n");

      c1 = 'Z';
      printf("\n2. isupper test: ");
      if(isupper(c1))
            printf("\nUPPER CASE\n");
      else if (islower(c1))
            printf("\nLOWER CASE\n");
      else
            printf("\nwoops 2!\n");

      if(isalpha(c1))
            printf("\nCharacter is alpha\n");
      else
            printf("\nCharacter is NOT alpha\n");

      c1 = '3';
      if(isdigit(c1))
            printf("\nCharacter is digit\n");
      else
            printf("\nCharacter is NOT digit\n");

      c1 = '3';
      if(isalnum(c1))
            printf("\nCharacter is alnum\n");
```

```
      else
            printf("\nCharacter is NOT alnum\n");


      c1 = 'd';
      if(isalnum(c1))
            printf("\nCharacter is alnum\n");
      else
            printf("\nCharacter is NOT alnum\n");


      c1 = ' ';
      if(isalnum(c1))
            printf("\nCharacter is alnum\n");
      else
            printf("\nCharacter is NOT alnum\n");

      c1 = '^';
      if(isalnum(c1))
            printf("\nCharacter is alnum\n");
      else
            printf("\nCharacter is NOT alnum\n");


      c1 = '^';
      if(isspace(c1))
            printf("\nCharacter is space\n");
      else
            printf("\nCharacter is NOT space\n");


      c1 = ' ';
      if(isspace(c1))
            printf("\nCharacter is space\n");
      else
            printf("\nCharacter is NOT space\n");

      return 0;
}
```


"C:\Documents and Settings\JKR\Debug\char_io_test.exe"
d
dAa
1. isupper test:
LOWER CASE
2. isupper test:
UPPER CASE
Character is alpha
Character is digit
Character is alnum
Character is alnum
Character is NOT alnum
Character is NOT alnum
Character is NOT space
Character is space
Press any key to continue