Problem Solving with C

Compiled by

M S Anand (anandms@pes.edu)

Text Book(s):

1. “How To Solve It By Computer”, R G Dromey, Pearson, 2011.
2. “The C Programming Language”, Brian Kernighan, Dennis Ritchie, 2nd Edition, Prentice Hall PTR, 1988.

Reference Book(s):

1. “Expert C Programming; Deep C secrets”, Peter van der Linden
2. “ The C puzzle Book”, Alan R Feuer

Arrays

An array is a fixed number of data items that **are all of the same type**. The data items in an array are referred to as elements. The elements in an array are all of type int, or of type long, or all of any type you choose.

```
char   c_array [10];
short    s_array [20];
```

Access the elements through the index which starts from 0.

An array is defined as **finite ordered collection of homogenous** data, stored in contiguous memory locations.
finite *means* data range must be defined.
ordered *means* data must be stored in continuous memory addresses.
homogenous *means* data must be of similar data type.

Initialization of an Array

After an array is declared it must be initialized. Otherwise, it will contain **garbage** value(any random value). An array can be initialized at either **compile time** or at **runtime**.

Compile time Array initialization

Compile time initialization of array elements is same as ordinary variable initialization. The general form of initialization of array is,

```
data-type array-name[size] = { list of values };
int marks[4]={ 67, 87, 56, 77 }; // integer array initialization
float area[5]={ 23.4, 6.8, 5.5 }; // float array initialization
int arr[] = {2, 3, 4};
int marks[4]={ 67, 87, 56, 77, 59 }; // Compile time error
```

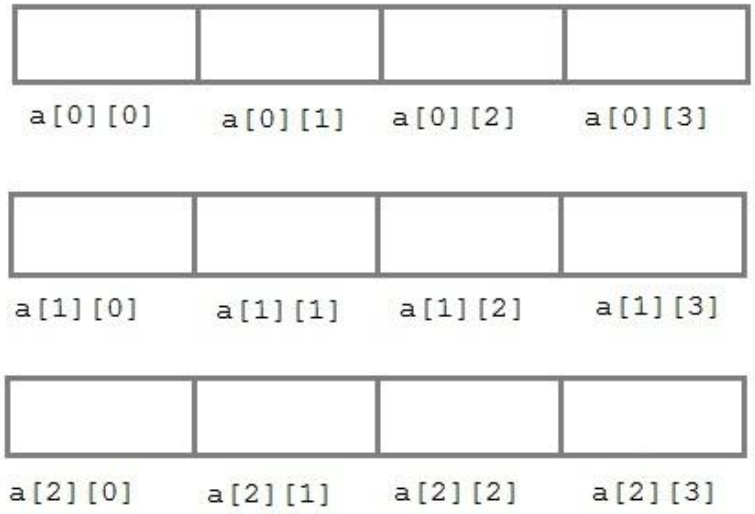
Runtime initialization
Use scanf to get the input into the array;

Multi-dimensional arrays

data-type array-name[row-size][column-size]

/* Example */

int a[3][4];



Declaring and initializing an array together

```
int arr[][3] = {
        {0,0,0},
        {1,1,1}
        };
```

Note: We have not assigned any row value to our array in the above example. It means we can initialize any number of rows. But, we must always specify number of columns, else it will give a compile time error.

Runtime initialization of a two dimensional Array

```
#include<stdio.h>
int main(void)
{
        int arr[3][4];
        int i, j;
        printf("Enter array elements");
        for(i = 0; i < 3;i++)
                for(j = 0; j < 4; j++)
                        scanf("%d", &arr[i][j]);

        for(i = 0; i < 3; i++)
                for(j = 0; j < 4; j++)
                        printf("%d", arr[i][j]);
        return 0;
}
```

Row-major and column-major methods of storing

In computing, **row-major order** and **column-major order** are methods for storing multidimensional arrays in linear storage such as random access memory.

The difference between the orders lies in which elements of an array are contiguous in memory. In a row-major order, the consecutive elements of a row reside next to each other, whereas the same holds true for consecutive elements of a column in a column-major order.

Programming languages or their standard libraries that support multi-dimensional arrays typically have a native row-major or column-major storage order for these arrays.

Row-major order is used in C/C++/Objective-C (for C-style arrays), etc
Column-major order is used in Fortran, MATLAB, etc

String and Character Array

String is a sequence of characters that is treated as a single data item and terminated by null character '\0'. Remember that C language does not support strings as a data type.

A **string** is actually one-dimensional array of characters in C language. These are often used to create meaningful and readable programs.

For example: The string "hello world" contains 12 characters including '\0' character which is automatically added by the compiler at the end of the string.

Declaring and Initializing a string variables

char name[13] = "StudyTonight"; // valid character array initialization

char name[10] = {'L','e','s','s','o','n','s','\0'}; // valid initialization

Remember that when you initialize a character array by listing all of its characters separately then you must supply the '\0' character explicitly.

Some examples of illegal initialization of character array are,

char ch[3] = "hell"; // Illegal

char str[4];

str = "hell"; // Illegal

String Input and Output

Input function scanf() can be used with **%s** format specifier to read a string input from the terminal. But there is one problem with scanf() function, it terminates its input on the first white space it encounters. Therefore if you try to read an input string "Hello World“ using scanf() function, it will only read **Hello** and terminate after encountering white spaces.

However, C supports a format specification known as the **edit set conversion code %[..]** that can be used to read a line containing a variety of characters, including white spaces.

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

int main(void)
{
        char str[20];
        printf("Enter a string");
        scanf("%[^\n]", str);
//scanning the whole string, including the white spaces
        printf("%s", str);
        return 0;
}
```

Write functions to perform the following:

1. Concatenating two strings
2. Finding the length of a string
3. Reversing a string
4. Copying one string to another
5. Comparing two strings

Hands-on session

Programs

Write a program in C to simulate a calculator with the following operations: +, -, * and /.

Write a program in C to print the tables from 1 to 12 in a format shown below:

Write a program in C to convert a given string (for ex “1234”) to the corresponding integer.

Write a program to convert a given integer value to a string format.

Write a program to convert a given integer into a string which has the binary equivalent of the number.
Example: Assuming that an integer is 32-bits long and for the number 5, the resultant string should be
“00000000000000000000000000000101”

Variable Length Arrays (C99)

We can define arrays where the dimensions are determined at runtime, when you execute the program.

```
int size = 0;
printf("Enter the number of elements you want to store: ");
scanf("%d", &size);
float values[size];
```

You can also define arrays with two or more dimensions with any or all of the dimensions determined at execution time. For example:

```
int rows = 0;
int columns = 0;
printf("Enter the number of rows you want to store: ");
scanf("%d", &rows);
printf("Enter the number of columns in a row: ");
scanf("%d", &columns);
float beans[rows][columns];
```

A C11-conforming compiler does not have to implement support for variable length arrays because it is an optional feature. If it does not, the symbol __STDC_NO_VLA__ must be defined as 1. You can check for support for variable length arrays using this code:

```
#ifdef __STDC_NO_VLA__
printf("Variable length arrays are not supported.\n");
exit(1);
#endif
```

String

A *string constant is a sequence of characters or symbols* between a pair of double-quote characters. Anything between a pair of double quotes is interpreted by the compiler as a string, including any special characters and embedded spaces.

Variables that store Strings

C has no specific provision within its syntax for variables that store strings, and because there are no string variable types, there are no special operators in the language for processing strings. This is not a problem though, because the standard library provides an extensive range of functions to handle strings.

We use an array of type char to hold strings. This is the simplest form of string variable. You can declare an array variable like this:

```
char saying[20];
```

This variable can accommodate a string that contains up to 19 characters, because you must allow one element for the termination character. Of course, you could also use this array to store 20 characters that are not a string.

Caution

Remember that when you specify the dimension of an array that you intend to use to store a string, it must be at least one greater than the number of characters in the string that you want to store. The compiler automatically adds \0 to the end of every string constant.

You can initialize a string variable when you declare it:
char saying[] = "This is a string.";

Initializing a char array and declaring it as constant is a good way of handling standard messages:
const char message[] = "The end of the world is nigh.";

Because you declare message as const, it's protected from being modified explicitly within the program. Any attempt to do so will result in an error message from the compiler. This technique for defining standard messages is particularly useful if they're used in many places within a program. It prevents accidental modification of such constants in other parts of the program. Of course, if you do need to be able to change the message, then you shouldn't specify the array as const.

Arrays of Strings

You can use a two-dimensional array of elements of type char to store strings, where each row holds a separate string. In this way you can store a whole bunch of strings and refer to any of them through a single variable name, as in this example:

```
char sayings[3][32] = {
"Manners maketh man.",
"Many hands make light work.",
"Too many cooks spoil the broth."
};
```

This definition creates an array of three rows of 32 characters. The strings between the braces will be assigned in sequence to the three rows of the array, sayings[0], sayings[1], and sayings[2]. Note that you don’t need to put braces around each string. The compiler deduces that each string is intended to initialize one row of the array. The first dimension specifies the number of strings that the array can store. The second dimension is specified as 32, which is just sufficient to accommodate the longest string, including its terminating \0 character.

The Idea of a Pointer

C provides a remarkably useful type of variable called a pointer. A *pointer* is a variable that stores an address—that is, its value is the address of another location in memory that can contain a value.

You already used an address when you used the scanf() function. A pointer variable with the name pNumber is defined by the second of the following two statements:

```
int Number = 25;
int *pNumber = &Number;
```

You declare a variable, Number, with the value 25, and a pointer, pNumber, which contains the address of Number.

You can now use the variable pNumber in the expression *pNumber to obtain the value contained in Number. The * is the dereference operator, and its effect is to access the data stored at the address specified by a pointer.

Operations with Strings

The standard library provides a number of functions for processing strings. To use these, you must include the string.h header file into your source file. String functions introduced in the C11 standard are safer and more robust than the traditional functions you may be used to using. They offer greater protection against errors such as buffer overflow. However, that protection is dependent on careful and correct coding. These string processing functions do bounds checking for arrays. The names of these functions end with _s.

How do you know whether your compiler supports these functions?

```
#include <stdio.h>
int main(void)
{
#if defined __STDC_LIB_EXT1__
printf("Optional functions are defined.\n");
#else
printf("Optional functions are not defined.\n");
#endif
return 0;
}
```

The nine most commonly used functions in the string library are:

strcat - concatenate two strings
strchr - string scanning operation
strcmp - compare two strings
strcpy - copy a string
strlen - get string length
strncat - concatenate one string with part of another
strncmp - compare parts of two strings
strncpy - copy part of a string
strrchr - a pointer to the **last** occurrence of c within s instead of the first.

The strcat function

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

The strcat() function appends a copy of the string pointed to by s2 (including the terminating null byte) to the end of the string pointed to by s1. The initial byte of s2 overwrites the null byte at the end of s1. If copying takes place between objects that overlap, the behavior is undefined. The function returns s1.

This function is used to attach one string to the end of another string.

The strchr function

char *strchr(const char *s, int c);

The strchr() function shall locate the first occurrence of c (converted to a char) in the string pointed to by s. The terminating null byte is considered to be part of the string.

The function returns the location of the found character, or a null pointer if the character was not found.

This function is used to find certain characters in strings.

The strcmp function

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

The strcmp() function shall compare the string pointed to by s1 to the string pointed to by s2. The sign of a non-zero return value shall be determined by the sign of the difference between the values of the first pair of bytes (both interpreted as type unsigned char) that differ in the strings being compared. Upon completion, strcmp() shall return an integer greater than, equal to, or less than 0, if the string pointed to by s1 is greater than, equal to, or less than the string pointed to by s2, respectively.

The strcpy function

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

The strcpy() function shall copy the C string pointed to by s2 (including the terminating null byte) into the array pointed to by s1. If copying takes place between objects that overlap, the behavior is undefined. The function returns s1. There is no value used to indicate an error: if the arguments to strcpy() are correct, and the destination buffer is large enough, the function will never fail.

Important: You must ensure that the destination buffer (s1) is able to contain all the characters in the source array, including the terminating null byte. Otherwise, strcpy() will overwrite memory past the end of the buffer, causing a buffer overflow, which can cause the program to crash, or can be exploited by hackers to compromise the security of the computer.

The strlen function

size_t strlen(const char *s);

The strlen() function shall compute the number of bytes in the string to which s points, not including the terminating null byte. It returns the number of bytes in the string. No value is used to indicate an error.

The strncat function

char *strncat(char *s1, const char *s2, size_t n);

The strncat() function shall append not more than n bytes (a null byte and bytes that follow it are not appended) from the array pointed to by s2 to the end of the string pointed to by s1. The initial byte of s2 overwrites the null byte at the end of s1. A terminating null byte is always appended to the result. If copying takes place between objects that overlap, the behavior is undefined. The function returns s1

The strncmp function

int strncmp(const char *s1, const char *s2, size_t n);

The strncmp() function shall compare not more than n bytes (bytes that follow a null byte are not compared) from the array pointed to by s1 to the array pointed to by s2. The sign of a non-zero return value is determined by the sign of the difference between the values of the first pair of bytes (both interpreted as type unsigned char) that differ in the strings being compared. See strcmp for an explanation of the return value.

The strncpy function

char *strncpy(char *s1, const char *s2, size_t n);

The strncpy() function shall copy not more than n bytes (bytes that follow a null byte are not copied) from the array pointed to by s2 to the array pointed to by s1. If copying takes place between objects that overlap, the behavior is undefined. If the array pointed to by s2 is a string that is shorter than n bytes, null bytes shall be appended to the copy in the array pointed to by s1, until n bytes in all are written. The function shall return s1; no return value is reserved to indicate an error.

It is possible that the function will **not** return a null-terminated string, which happens if the s2 string is longer than n bytes.

The strrchr function

char *strrchr(const char *s, int c);

The strrchr function is similar to the strchr function, except that strrchr returns a pointer to the **last** occurrence of c within s instead of the first.

The strrchr() function shall locate the last occurrence of c (converted to a char) in the string pointed to by s. The terminating null byte is considered to be part of the string. Its return value is similar to strchr's return value.

Summary

char *strcat(char * s1, const char * s2);
char *strchr(const char *s, int c);
int strcmp(const char *s1, const char *s2);
char *strcpy(char *s1, const char *s2);
int strlen(const char *s);
char *strncat(char *s1, const char *s2, int n);
int strncmp(const char *s1, const char *s2, int n);
char *strncpy(char *s1, const char *s2, int n);
char *strrchr(const char *s, int c);