C LANGUAGE

Kalpesh Chauhan

Arrays

- Arrays a kind of data structure that can store a fixed-size sequential collection of elements of the same type. An array is used to store a collection of data, but it is often more useful to think of an array as a collection of variables of the same type.

- Instead of declaring individual variables, such as number0, number1, ..., and number99, you declare one array variable such as numbers and use numbers[0], numbers[1], and ..., numbers[99] to represent individual variables. A specific element in an array is accessed by an index.
- All arrays consist of contiguous memory locations. The lowest address corresponds to the first element and the highest address to the last element.

First					Last
Number[0]	Number[1]	Number[2]	Number[3]	Number[4]	Number[5]

Declaring Arrays

- To declare an array in C, a programmer specifies the type of the elements and the number of elements required by an array as follows

- Type array_name [Size]

- This is called a *single-dimensional* array. The **arraySize** must be an integer constant greater than zero and **type** can be any valid C data type. For example, to declare a 10-element array called **balance** of type double, use this statement.
- double balance[10];
- Here *balance* is a variable array which is sufficient to hold up to 10 double numbers.

Initializing Arrays

- You can initialize an array in C either one by one or using a single statement as follows.

- double balance[5] = {1000.0, 2.0, 3.4, 7.0, 50.0};

- The number of values between braces { } cannot be larger than the number of elements that we declare for the array
- If you omit the size of the array, an array just big enough to hold the initialization is created. Therefore, if you write between square brackets [ ].
- double balance[] = {1000.0, 2.0, 3.4, 7.0, 50.0};

- You will create exactly the same array as you did in the previous example. Following is an example to assign a single element of the array.

- balance[4] = 50.0;

- The above statement assigns the 5th element in the array with a value of 50.0. All arrays have 0 as the index of their first element which is also called the base index and the last index of an array will be total size of the array minus 1. the array index was started from zero(0).

Two Dimension Array

- The simplest form of multidimensional array is the two-dimensional array. A two-dimensional array is, in essence, a list of one-dimensional arrays. To declare a two-dimensional integer array of size [x][y], you would write something as follows,

- Type name_array [x][y];

  Ex. int balance[3][3]

Balance	0	1	2
0			
1		500	
2			

```
Balance [1][1] = 500;
```

Three dimension array

- The three dimension array collection of rows and columns.

- To declare three dimension array use following syntax

```
Type  array_name [size][rows][columns]

int ip[2][2][2]
```

* Above statement creates array of 8 elements

int bal[3][3][3]

	0					1				2		
	0	1	2			0	1	2		0	1	2
0	100				0	200				0		
1					1					1		
2					2					2		

Bal[0][0][0] = 1oo

Bal[1][0][0] = 200

Character array

- Also you can create array of characters, its known as string.

- Ex.
  - Char name[10] = “rajkot”;

String must be specified in Double quote only(“”),

In the above example index of **r** is 0, **a** is 1 and so on.

Functions

Functions

- A function is a group of statements that together perform a task. Every C program has at least one function, which is **main()**, and all the most trivial programs can define additional functions.
- You can divide up your code into separate functions. How you divide up your code among different functions is up to you, but logically the division is such that each function performs a specific task

- A function **declaration** tells the compiler about a function's name, return type, and parameters. A function **definition** provides the actual body of the function.
- A function can also be referred as a method or a sub-routine or a procedure, etc.

Type of Functions

- User Define Functions (UDF)
- In Built Functions

User Define Functions

- When a programmer want to create new own block of code its known as user define functions.

- Types of UDF is
  - No parameters no return value
  - No parameters with return value
  - With parameters no return
  - With parameters with return

Syntax of function

```
return_type function_name( parameter list )
{
    body of the function
}
```

- **Return Type** − A function may return a value. The **return_type** is the data type of the value the function returns. Some functions perform the desired operations without returning a value. In this case, the return_type is the keyword **void**.

- **Function Name** − This is the actual name of the function. The function name and the parameter list together constitute the function signature.

- **Parameters** − A parameter is like a placeholder. When a function is invoked, you pass a value to the parameter. This value is referred to as actual parameter or argument. The parameter list refers to the type, order, and number of the parameters of a function. Parameters are optional; that is, a function may contain no parameters.
- **Function Body** − The function body contains a collection of statements that define what the function does.

Main parts of functions

- Function Declarations
- Calling a Function
- Function definition

Function declaration

- A function **declaration** tells the compiler about a function name and how to call the function. The actual body of the function can be defined separately.

- return_type function_name(parameter list);

- Ex   void max(int I, int j);

Function calling

- While creating a C function, you give a definition of what the function has to do. To use a function, you will have to call that function to perform the defined task.

- Ex   max(10,16);

Function definition

```
Void max(int I, int j)
{
    If(i>j)
    {
          printf("%d is greater ",i);
    }
    else
    {
       printf("%d is greater ",j);
    }
}
```

Built in functions

- There is a rich collection of built in functions in c environment.

- This function divided in several header files like. string, ctype, math, dos, stdlib and etc..

String functions

strlen()	Calculates the length of string
strcpy()	Copies a string to another string
strcat()	Concatenates(joins) two strings
strcmp()	Compares two string
strlwr()	Converts string to lowercase
strupr()	Converts string to uppercase

Character functions

int isalpha(c);	c is a letter.
int isupper(c);	c is an upper case letter.
int islower(c);	c is a lower case letter.
int isdigit(c);	c is a digit [0-9].
int isalnum(c);	c is an alphanumeric
int isspace(c);	c is a SPACE
int ispunct(c);	c is a punctuation character

getch()	get single character from keyboard but not print it.
getche()	get single character from key board and print while scanning.

Math functions

ceil()	Returns nearest integer greater than argument passed.
exp()	Computes the e raised to given power.
floor()	Returns nearest integer lower than the argument passed.
pow()	Computes the number raised to given power.
sqrt()	Computes square root of the argument.

User defined functions

- **Benefits of Using Functions**

1. It provides modularity to the program.
2. Easy code Re-useability. You just have to call the function by its name to use it.
3. In case of large programs with thousands of code lines, debugging and editing becomes easier if you use functions.

Function syntax

```
return-type function-name (parameter-list)
{
        function-body ;
}
```

Structure / Union

Introduction

- Arrays allow to define type of variables that can hold several data items of the same kind. Similarly **structure** is another user defined data type available in C that allows to combine data items of different kinds.

How to Defining a Structure?

- To define a structure, you must use the **struct** statement. The struct statement defines a new data type, with more than one member. The format of the struct statement is as follows −

```
struct [structure tag]
{
   member definition;
   member definition;
} [one or more structure variables];
```

Accessing Structure Members

- To access any member of a structure, we use the **member access operator** (.). The member access operator is coded as a period between the structure variable name and the structure member that we wish to access. You would use the keyword **struct** to define variables of structure type.

Union

- A **union** is a special data type available in **C** that allows to store different data types in the same memory location. You can define a **union** with many members, but only one member can contain a value at any given time. **Unions** provide an efficient way of using the same memory location for multiple-purpose.

Typedef keyword

- The **Typedef** Keyword in **C** and C++ The **typedef** keyword allows the programmer to create new names for **types** such as int or, more commonly in C++, templated **types**--it literally stands for "**type** definition".

Variable Scope and Storage Class

Scope of variables

- A scope in any programming is a region of the program where a defined variable can have its existence and beyond that variable it cannot be accessed. There are three places where variables can be declared in C programming language.

1. Inside a function or a block which is called **local** variables.
2. Outside of all functions which is called **global** variables.
3. In the definition of function parameters which are called **formal** parameters.

Local Variables

- Variables that are declared inside a function or block are called local variables. They can be used only by statements that are inside that function or block of code. Local variables are not known to functions outside their own.

Global Variables

- Global variables are defined outside a function, usually on top of the program. Global variables hold their values throughout the lifetime of your program and they can be accessed inside any of the functions defined for the program.
- A global variable can be accessed by any function. That is, a global variable is available for use throughout your entire program after its declaration.

Formal Parameters

- Formal parameters, are treated as local variables with-in a function and they take precedence over global variables.

Header Files

- A header file is a file with extension **.h** which contains C function declarations and macro definitions to be shared between several source files. There are two types of header files: the files that the programmer writes and the files that comes with your compiler.
- You request to use a header file in your program by including it with the C preprocessing directive **#include**, like you have seen inclusion of **stdio.h** header file, which comes along with your compiler.

- Including a header file is equal to copying the content of the header file but we do not do it because it will be error-prone and it is not a good idea to copy the content of a header file in the source files, especially if we have multiple source files in a program.

Include Syntax

- Both the user and the system header files are included using the preprocessing directive **#include.**

- System header files
  - #include <stdio.h>
- User header files
  - #include “stdio.h”

C Storage Class

- A storage class defines the scope (visibility) and life-time of variables and/or functions within a C Program. They precede the type that they modify. We have four different storage classes in a C program −

1. auto
2. register
3. static
4. extern

The auto Storage Class

- The **auto** storage class is the default storage class for all local variables.

- { int mount; auto int month; }

- The example above defines two variables with in the same storage class. 'auto' can only be used within functions, i.e., local variables.

The register Storage Class

- The **register** storage class is used to define local variables that should be stored in a register instead of RAM.

  { register int miles; }

- The register should only be used for variables that require quick access such as counters. It should also be noted that defining 'register' does not mean that the variable will be stored in a register. It means that it MIGHT be stored in a register depending on hardware and implementation restrictions.

The static Storage Class

- The **static** storage class instructs the compiler to keep a local variable in existence during the life-time of the program instead of creating and destroying it each time it comes into and goes out of scope. Therefore, making local variables static allows them to maintain their values between function calls.

- he static modifier may also be applied to global variables. When this is done, it causes that variable's scope to be restricted to the file in which it is declared.
- In C programming, when **static** is used on a class data member, it causes only one copy of that member to be shared by all the objects of its class.

The Extern Storage Class

- Extern storage class simply tells us that the variable is defined elsewhere and not within the same block where it is used. Basically, the value is assigned to it in a different block and this can be overwritten/changed in a different block as well. So an extern variable is nothing but a global variable initialized with a legal value where it is declared in order to be used elsewhere.