1. C is a general purpose, structured language. its instructions consist of terms that resembles algebraic expressions, augmented by certain English keywords such as if, else, for and while. C was the offspring of the ‘Basic Combined Programming Language’(BPCL) called B, developed in the 1960s’s at Cambridge University.

C is characterized by the ability to write very concise source programs, due in part of to the large number of operators included within the language.

It has relatively small instructions, set though actual implementations include extensively library functions which enhance the basic instructions.

The language encourages users to write additional library functions of their own. Thus, the features and capabilities of the language can easily be extended by the user.

C compilers are commonly used for computers of all sizes. The compilers are usually compact, and they generate object programs that are small and highly efficient when compared with programs compiled from other high-level languages.

The programs are highly portable, even more so than with other high-level languages. The reason for this is that C relegates most computer dependent features to its library functions.

A program can be viewed as a group of building blocks called functions called functions. A function is a subroutine that may include one or more statements designed to perform a special task.

Every C program must have one main function sections. This section contains two parts, declaration part and executable part. The declaration part declares all the variables used in the executable part. These two parts must appear between opening and closing braces {(and)}. The program execution begins at the opening brace and ends at the closing brace. The closing section of the main function section is the logical end of the program. All state declaration and executable parts end with a semicolon (;)

#include<stdio.h>

Int main () {

float r;

printf(“enter the radius”);

scanf(“%f”,&radius);

printf(“Area of circle %f“, 3.14\*radius\*radius);

return 0;

}

1. Every statement in the computer is executed based on pre-defined rules. The control flow is also based on logic. At times you find a necessity to execute a few customized logics. Custom statements can be executed using control statements.

Control statements enters the statement block and gets executed if the logic is satisfied. They are often used to determine the order in which statements must be executed. Control statements in C help the computer execute a certain logical statement or not. It is used to direct the execution of statements under certain conditions.

Types of control statements in C

* Decision- making control statements.
* Conditional statements
* Go to statements in C
* Loop control statements in C

Decision – making control statements;

* Simple if statement
* If-else statement
* Nested if-else statements
* Else -if ladder
* Simple if statement

Simple if statements are carried out to perform some operation when the condition is only true. If the condition of the statement of the if statement is true then the statement under the if block is executed else the control is transferred to the statements outside the if block.

Syntax of the if statement is as given below;

If (condition)

(

Statement1;

)

* If-else statement

In some situations, you may have to execute statements based on true or false under certain conditions, therefore; you use if-else statements the condition is true, then if block will be executed otherwise the else block is executed.

Syntax of the if-else statement is as given below;

If (condition)

Statement1;

Else

Statement2;

* Nested -if statement

The nested if-else statements consist of another if or else. Therefore; if the condition of “if” is true then outer if’s if block is executed which contains another if and if the conditions of if block is true, statements of inners if’s “else” block will be executed.

If the outer “if “condition is not true then the outer if’s “else” block is executed which consists of another if. The outer else’s inner if the condition is true then the statement under outer else’s inner if is executed else the outer else’s else block is executed.

1. Storage classes in C

C storage classes are used to describe the features of a variable/function. These features basically include the scope, visibility and lifetime which help us to trace the existence of a particular variable during the runtime of a program.

C languages uses 4 storage classes ;

1. Auto

This is the default storage class for all the variables declared inside a function or a block. Hence, the keyword auto is rarely used while writing programs in C language. Auto variables can only be accessed within the block /function they have been declared and not outside them. Of course, these can be accessed within nested blocks within the parent block/function in which the auto variable was declared.

However, they can be accessed outside their scope as well using the concept of pointers given by pointing to the very exact memory location where the variables reside. They are assigned a garbage value by default whenever they are declared.

1. Extern

Extern storage class simply tell 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 initialized with a legal value where it is declared in order to be used elsewhere. it can be accessed within any function/block.

Also, normal global variable can be made extern as well by placing the ‘extern’ keyword before its declaration /definition in any function/block. This basically signifies that we are not initializing a new variable but instead, we are using /accessing the global variable only. The main purpose of using extern variable is that they can be accessed between two different files which are part of a large program.

1. Static

This storage class is used to declare static variables which are popularly used while writing programs in C language. Static variables have the property of preserving their value even after they are out of their scope! Hence static variables preserve the value of their last use in their scope. So, we can say that they are initialized only once and exist till the termination of the program. Thus, no new memory is located because they are not redeclared.

Their scope is local to the function to which they were defined. Global static variables can be accessed anywhere in the program. By default, they are assigned the value 0 by the compiler.

1. Register

This storage class declares register variables that have the same functionality as that of the auto variables. The only difference is that the compiler tries to store these variables in the register of the microprocessor if a free register is available. this makes the use of the register variables to be much faster than that of the variables stored in the memory during the runtime of the program.

Syntax

To specify the storage class for a variable, the following syntax is to be followed;

storage \_classvariable datatype var\_ name;

1. Array can be defined as a method of clubbing multiple entities of similar type into a larger group. These entities or elements can be of float, char, or double data type etc. It is a simple and fast way of storing multiple values under a single name.

Array declaration

In C, we have to declare the way the array like any other variable before using it. We can declare an array by specifying its name, the type of its elements, and the size of its dimensions. When we declare an array in C, the compiler allocates the memory block of the specified size of the array name.

Syntax of Array declaration

data\_ type array \_ name [size];

Or

data\_ type array \_ name [size1] [size2] …… [size n]

where n is the number of dimensions.

Example of array declaration;

#include <stdio.h>

Int main ()

{

// declaring array of integer type

Int arr\_\_int [5];

// declaring array of character

Char arr\_\_char[5];

return 0;

C array Initialization

Initialization in C is the process to assign some initial value to the variable. When the array is declared or allocated memory, the elements of the array contain some garbage value. So, we need to initialize the array to some meaningful value. There are multiple ways in which we can initialize an array in C.

1. Array initialization with declaration

In this method, we initialize the array along with its declaration. We use an initializer list to initialize multiple elements of the array. An initializer list is the list of values enclosed within {} separated by a comma.

1. Array initialization with declaration without size

If we initialize an array using an initializer list, we can skip declaring the size of the array in these cases. The size of the array is equal to the number of elements present in the initializer list as the compiler can automatically deduce the size of the array.

1. Array declaration after declaration

Initialize the array after the declaration by assigning the initial value to each element individually. We can use for loop, while loop, or do-while loop to assign the value to each element of the array.

Example of array initialization in C

#include <stdio.h>

Int main ()

{

// array initialization using initializer list

Int ar[5]={10,20,30,40,50};

// array initialization using initializer list without

// specifying size

Int arr1[] ={1,2,3,4,5};

// array initialization using for loop

Float arr2[5];

for(int i=0; i<5; i++) {

arr2[i] = (float ) i\*2.1;

}

return 0;

}

1. (a) A pointer is the variable used to store memory address. Hence, we have to declare and initialize it just like any other variable. Every pointer variable has a data type associated with it, which means an integer pointer can hold only integer variable addresses.

Declaration of pointer variable

* Data- type is a valid C data type.
* \* Symbol specifies it is a pointer variable. You must prefix \* before variable name to declare it as a pointer.
* Pointer – variable – name is a valid C identifier i.e., the name of pointer variable.

Eg. - int \* ptr;

Pointer Initialization

Pointer initialization is the process where we assign some initial value to the pointer variable. General use the (&) address of operator to get the memory address of a variable and then store it in the pointer variable.

Eg - int var = 10;

Int \* ptr;

ptr=&var;

Initialize the pointer in a single step. This method is called pointer definition as the pointer is declared and initialized at the same time.

Eg – int \*ptr=&var;

1. Once the pointer variable points to an object , we can use the indirection operator [\*] to access what’s stored in the object.

The following below has the programing lines to understand the concept of indirect operator.

Eg- #include <stdio.h>

Int main () {

int a;

a = 10;

int \*p;

p = &a;

printf(“%d” , a);

printf(“%d”,\*p);

\*p = 2;

printf(“%d”,\*p);

return 0;

}

Generally , w can access the values of a variable as shown in the fifth line.

In C , we can access the value to which the pointer variable is pointing. This kind of access is done by using the indirect operator.

The sixth line of the code is how we access the value stored in a variable. Not only to access but to modify or update, we can use the indirection operator.

The seventh line of code shows how to modify the value by using the indirection operator.

The structure in C is a user- defined data type that can be used to group items of possibly different types into a single type. The struct keyword is used to define the structure in the C programming language. The items in the structure are filled its member and they can be of any valid data type.

C Structure Declaration

To declare structure in C before using in it in our program. In structure declaration, we specify its member variables along with their datatype. We can use the struct keyword to declare the structure in C using the following syntax;

Syntax

Struct structure \_name {

datatype member\_name1;

data\_ type member \_name1;

……………………….

…………………………….

}

The above syntax is also called a structure template or structure prototype and no memory is allocated to the structure in the declaration.

Structure Variable Declaration with Structure Template;

Struct structure name {

Datatype member \_name1;

Datatype member \_name1;

………………..

……………….

} variable1, variable2, …….

Structure Variable Declaration after structure Template

// structure declared before

struct structure \_name variable1, variable2.

Access Structure Members

We can access structure members by using the (.) dot operator

Syntax

Structure name. member1;

Structure name. member 2;

In the case where we have a pointer to the structure, we can also use the arrow operator to access the members.

(b)

#include <stdio.h>

#include <string.h>

// Define the structure for employee details

struct Employee {

char name[50];

int age;

float salary;

};

int main() {

// Create an array of structures to store multiple employee details

struct Employee employees[5];

// Input employee details

for (int i = 0; i < 5; i++) {

printf("Enter details for Employee %d:\n", i + 1);

printf("Name: ");

scanf("%s", employees[i].name);

printf("Age: ");

scanf("%d", &employees[i].age);

printf("Salary: ");

scanf("%f", &employees[i].salary);

printf("\n");

}

// Display employee details

printf("\nEmployee Details:\n");

for (int i = 0; i < 5; i++) {

printf("Employee %d\n", i + 1);

printf("Name: %s\n", employees[i].name);

printf("Age: %d\n", employees[i].age);

printf("Salary: %.2f\n", employees[i].salary);

printf("\n");

}

return 0;

}