An Industrial Training Report on

**“C-LANGUAGE”**

Submitted in partial fulfilment for the award

of the Degree of

**Bachelor of Technology**

in

**Department of Computer Science & Engineering**



**Submitted To:** **Submitted By:**

Mr. Shashi Kant Raza Rajkumar Jangid

Assistant Prof. (CSE Dept.) 20EMCCS101

**Department of Computer Science & Engineering**

Modern Institute of Technology & Research Centre

Bikaner Technical University, Bikaner

**DECEMBER, 2021**

**ACKNOWLEDGEMENT**

Firstly, I would like to express my gratitude to my advisor for the beneficial

Comments and remarks.

I express my sincere thanks to **Prof. S. K. Sharma (Director), Modern**

**Institute of Technology and Research Centre,Alwar and Prof. VikasThada,**

**Dean of Academics, Modern Institute of Technology and Research Centre,**

**Alwar.**

I pay my deep sense of gratitude to **Mr. Arvind Sharma (HOD) of CSE**

**department, Modern Institute of Technology and Research Centre, Alwar**

to encourage me to the highest peak and to provide me the opportunity to present the seminar.

I feel to acknowledge my indebtedness and deep sense of gratitude to my guide

**Mr. Shashi Kant Raza, Assistant Professor (CSE Department)** whose

valuable guidance and kind supervision given to me.

I am immensely obliged to my friends for their elevating inspiration, encouraging guidance and kind supervision.

Last, but not the least, my parents are also an important inspiration for me. So,

with due regards, I express my gratitude to them.

Rajkumar Jangid

20EMCCS101

# UNDERTAKING

I declare that the work presented in this report titled **“C-LANGUAGE”**, submitted to the **Computer Science and Engineering Department**, **Modern Institute of Technology and Research Centre, Alwar**, for the award of Bachelor of Technology degree in Computer Science and Engineering, is my original work. I have not plagiarized or submitted the same work for the award of any other degree. In case this undertaking is found incorrect, I accept my mistake.

**December, 2021 Rajkumar Jangid**

**20EMCCS101**

**Table of Content**

**UNDERTAKING………………………………………………………………………………ii**

**ACKNOWLEDGEMENT…………………………………………………………….………iii**

**UNIT-I………………………………………………………………………………………...2-15**

**Introduction to C**– History of C, Features of C, Structure of C Program, Character Set, C Tokens-Keywords, Identifiers, Constants, Variables, Data types, Operators.

**UNIT**‐**II………………………………………………………………………………………16-26**

**Statements**‐Selection statements (Decision Making)- if and switch statements with examples, Repetition statements (loops)- while, for, do‐while statements with examples, Unconditional statements- break, continue, goto statements with examples.

**UNIT – III……………………………………………………………………………………27-41**

**Functions**‐Designing Structured Programs, Types of Functions-User defined functions, Standard functions, Categories of functions, Parameter Passing techniques, Storage classes, Recursion.

**UNIT**‐**IV……………………………………………………………………………………..41-46**

**Arrays-** Declaration and Initialization, One dimensional Arrays.

**Strings-** Declaration and Initialization.

**UNIT-V………………………………………………………………………………………47-49**

**Pointers-** Introduction, Definition and Declaration of pointers, address operator, Pointer variables, Pointers with Arrays.

**TEXT BOOKS:**

1. Computer Programming with C, Special Edition‐MRCET, Mc Graw Hill Publishers 2017.
2. Computer Science: A Structured Programming Approach Using C, B.A.Forouzan and R.F. Gilberg, Third Edition, Cengage Learning.

**REFERENCE BOOKS:**

1. The C Programming Language, B.W. Kernighan and Dennis M.Ritchie, PHI.
2. Computer Programming, E.Balagurusamy, First Edition, TMH.
3. C and Data structures – P. Padmanabham, Third Edition, B.S. Publications.
4. Programming in *C, Ashok Kamthane*. Pearson Education India.
5. Let us C ,Yashwanth Kanethkar, 13th Edition, BPB Publications.

**Course Outcomes:**

* Demonstrate the basic knowledge of computer hardware and software.To formulate simple algorithms for arithmetic and logical problems.
* To translate the algorithms to programs (in C language).
* To test and execute the programs and correct syntax and logical errors.
* Ability to apply solving and logical skills to programming in C language and also in other languages.

**UNIT - I**

**Introduction to ‘C’ language**

**INTRODUCTION TO ‘C’ LANGUAGE:**

C language facilitates a very efficient approach to the development and implementation of computer programs. The History of C started in 1972 at the Bell Laboratories, USA where Dennis M. Ritchie proposed this language. In 1983 the American National Standards Institute (ANSI) established committee whose goal was to produce “an unambiguous and machine independent definition of the language C “ while still retaining it’s spirit .

C is the programming language most frequently associated with UNIX. Since the 1970s, the bulk of the UNIX operating system and its applications have been written in C. Because the C language does not directly rely on any specific hardware architecture, UNIX was one of the first portable operating systems. In other words, the majority of the code that makes up UNIX does not know and does not care which computer it is actually running on. Machine-specific features are isolated in a few modules within the UNIX kernel, which makes it easy for you to modify them when you are porting to a different hardware architecture.

C was first designed by Dennis Ritchie for use with UNIX on DEC PDP-11 computers. The language evolved from Martin Richard's BCPL, and one of its earlier forms was the B language, which was written by Ken Thompson for the DEC PDP-7. The first book on C was *The C Programming Language* by Brian Kernighan and Dennis Ritchie, published in 1978.

In 1983, the American National Standards Institute (ANSI) established a committee to standardize the definition of C. The resulting standard is known as *ANSI C,* and it is the recognized standard for the language, grammar, and a core set of libraries. The syntax is slightly different from the original C language, which is frequently called K&R for Kernighan and Ritchie. There is also an ISO (International Standards Organization) standard that is very similar to the ANSI standard.

It appears that there will be yet another ANSI C standard officially dated 1999 or in the early 2000 years; it is currently known as "C9X."

**BASIC STRUCTURE OF C LANGUAGE:**

The program written in C language follows this basic structure. The sequence of sections should be as they are in the basic structure. A C program should have one or more sections but the sequence of sections is to be followed.

1. Documentation section
2. Linking section
3. Definition section
4. Global declaration section
5. Main function section

{

Declaration section

Executable section

}

1. Sub program or function section

1. **DOCUMENTATION SECTION** **:** comes first and is used to document the use of logic or reasons in your program. It can be used to write the program's objective, developer and logic details. The documentation is done in C language with /\* and \*/ . Whatever is written between these two are called comments.
2. **LINKING SECTION :** This section tells the compiler to link the certain occurrences of keywords or functions in your program to the header files specified in this section.

e.g. #include <stdio.h>

1. **DEFINITION SECTION :** It is used to declare some constants and assign them some value.

e.g. #define MAX 25

Here #define is a compiler directive which tells the compiler whenever MAX is found in the program replace it with 25.

1. **GLOBAL DECLARATION SECTION :** Here the variables which are used through out the program (including main and other functions) are declared so as to make them global(i.e accessible to all parts of program)

e.g. int i; (before main())

1. **MAIN FUNCTION SECTION :** It tells the compiler where to start the execution from

main()

{

point from execution starts

}

main function has two sections

* + 1. declaration section : In this the variables and their data types are declared.
    2. Executable section : This has the part of program which actually performs the task we need.

1. **SUB PROGRAM OR FUNCTION SECTION :** This has all the sub programs or the functions which our program needs.

**SIMPLE ‘C’ PROGRAM:**

/\* simple program in c \*/

#include<stdio.h>

main()

{

printf(“welcome to c programming”);

} /\* End of main \*/

**C-TOKENS :**

Tokens are individual words and punctuations marks in English language sentence. The smallest individual units are known as C tokens.

OPERATORS

SPECIAL SYMBOLS

STRINGS

CONSTANTS

IDENTIFIERS

C TOKENS

E.g. +, -, \*

E.g. [ ], { }

E.g. “asifia”

E.g. -15.4, ‘a’, 200

|  |
| --- |
| KEY WORDS |

E.g. rate,no\_of\_hours

E.g. int, printf

A C program can be divided into these tokens. A C program contains minimum 3 c tokens no matter what the size of the program is.

**KEYWORDS :**

There are certain words, called keywords (reserved words) that have a predefined meaning in ‘C’ language. These keywords are only to be used for their intended purpose and not as identifiers.

The following table shows the standard ‘C’ keywords

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Auto | Break | Case | Char | Const | Continue |
| Default | Do | Double | Else | Enum | Extern |
| Float | For | Goto | If | Int | Long |
| Register | Return | Short | Signed | Sizeof | Static |
| Struct | Switch | Typedef | Union | Unsigned | void |
| Volatile | While |  |  |  |  |

**IDENTIFIERS :**

Names of the variables and other program elements such as functions, array,etc,are known as identifiers.

There are few rules that govern the way variable are named(identifiers).

* + 1. Identifiers can be named from the combination of A-Z, a-z, 0-9, \_(Underscore).
    2. The first alphabet of the identifier should be either an alphabet or an underscore. digit are not allowed.
    3. It should not be a keyword.

Eg: name,ptr,sum

After naming a variable we need to declare it to compiler of what data type it is .

The format of declaring a variable is

Data-type id1, id2,.....idn; where data type could be float, int, char or any of the data types.

id1, id2, id3 are the names of variable we use. In case of single variable no commas are required.

eg float a, b, c; int e, f, grand total;

char present\_or\_absent;

**ASSIGNING VALUES :**

When we name and declare variables we need to assign value to the variable. In some cases we assign value to the variable directly like

a=10;

in our program.

In some cases we need to assign values to variable after the user has given input for that.

eg we ask user to enter any no and input it

/\* write a program to show assigning of values to variables \*/

#include<stdio.h> main()

{

int a; float b;

printf("Enter any number\n");

b=190.5; scanf("%d",&a); printf("user entered %d", a); printf("B's values is %f", b);

}

**CONSTANTS :**

A quantity that does not vary during the execution of a program is known as a constant supports two types of constants namely Numeric constants and character constants.

**NUMERIC CONSTANTS:**

1. Example for an integer constant is 786,-127
2. Long constant is written with a terminal ‘l’or ‘L’,for example 1234567899L is a Long constant.
3. Unsigned constants are written with a terminal ‘u’ or ‘U’,and the suffix ‘ul’ and ‘UL’ indicates unsigned long. for example 123456789u is a Unsigned constant and

1234567891ul is an unsigned long constant.

1. The advantage of declaring an unsigned constant is to increase the range of storage.
2. Floating point constants contain a decimal point or an exponent or both. For Eg : 123.4,1e-2,1.4E-4,etc.The suffixes f or F indicate a float constant while the absence of f or F indicate the double, l or L indicate long double.

**CHARACTER CONSTANTS:**

A character constant is written as one character with in single quotes such as ‘a’. The value of a character constant is the numerical value of the character in the machines character set. certain character constants can be represented by escape sequences like ‘\n’. These sequences look like two characters but represent only one.

The following are the some of the examples of escape sequences:

**Escape sequence Description**

\a Alert

\b Backspace

\f Form feed

\n New Line

\r Carriage return

\t Horizontal Tab

\v Vertical Tab

String constants or string literal is a sequence of zero or more characters surrounded by a double quote. Example , “ I am a little boy”. quotes are not a part of the string.

To distinguish between a character constant and a string that contains a single character ex: ‘a’ is not same as “a”. ‘a’ is an integer used to produce the numeric value of letter a in the machine character set, while “a” is an array of characters containing one character and a ‘\0’ as a string in C is an array of characters terminated by NULL.

There is one another kind of constant i.e Enumeration constant , it is a list of constant integer values.

Ex.: enum color { RED, Green, BLUE }

The first name in the enum has the value 0 and the next 1 and so on unless explicit values are specified.

If not all values specified , unspecified values continue the progression from the last specified value. For example

Enum months { JAN=1, FEB,MAR, …, DEC -

Where the value of FEB is 2 and MAR is 3 and so on.

Enumerations provide a convenient way to associate constant values with names.

**VARIABLES :**

A quantity that can vary during the execution of a program is known as a variable. To identify a quantity we name the variable for example if we are calculating a sum of two numbers we will name the variable that will hold the value of sum of two numbers as 'sum'.

**DATA TYPES :**

To represent different types of data in C program we need different data types. A data type is essential to identify the storage representation and the type of operations that can be performed on that data. C supports four different classes of data types namely

1. Basic Data types
2. Derives data types
3. User defined data types
4. Pointer data types

**BASIC DATA TYPES:**

All arithmetic operations such as Addition , subtraction etc are possible on basic data types.

E.g.: int a,b;

Char c;

**The following table shows the Storage size and Range of basic data types:**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **TYPE** | **LENGTH** | | **RANGE** | |
| Unsigned char | 8 bits | | 0 to 255 | |
| Char | 8 bits | | -128 to 127 | |
| Short int | 16 bits | | -32768 to 32767 | |
| Unsigned int | 32 bits | | 0 to 4,294,967,295 | |
| Int |  | 32 bits |  | -2,147,483,648 to 2,147,483,648 | |
| Unsigned long |  | 32 bits |  | 0 to 4,294,967,295 | |
| Enum |  | 16 bits |  | -2,147,483,648 to 2,147,483,648 | |
| Long |  | 32 bits |  | -2,147,483,648 to 2,147,483,648 | |
| Float |  | 32 bits |  | 3.4\*10E-38 to 3.4\*10E38 | |
| Double |  | 64 bits |  | 1.7\*10E-308 to 1.7\*10E308 | |
| Long double |  | 80 bits |  | 3.4\*10E-4932 to 1.1\*10E4932 | |

**DERIVED DATA TYPES:**

Derived datatypes are used in ‘C’ to store a set of data values. Arrays and Structures are examples for derived data types.

Ex: int a[10];

Char name[20];

**USER DEFINED DATATYPES:**

C Provides a facility called typedef for creating new data type names defined by the user. For Example ,the declaration ,

**typedef int Integer;**

makes the name Integer a synonym of int.Now the type Integer can be used in declarations

,casts,etc,like,

**Integer num1,num2;**

Which will be treated by the C compiler as the declaration of num1,num2as int variables.

“typedef” ia more useful with structures and pointers.

**POINTER DATA TYPES:**

Pointer data type is necessary to store the address of a variable.

**INPUT AND OUTPUT STATEMENTS :**

The simplest of input operator is getchar to read a single character from the input device.

varname=getchar();

you need to declare varname.

The simplest of output operator is putchar to output a single character on the output device.

putchar(varname)

The getchar() is used only for one input and is not formatted. Formatted input refers to an input data that has been arranged in a particular format, for that we have scanf.

scanf("control string", arg1, arg2,...argn);

Control string specifies field format in which data is to be entered.

arg1, arg2... argn specifies address of location or variable where data is stored.

eg scanf("%d%d",&a,&b); %d used for integers

|  |  |  |
| --- | --- | --- |
| %f |  | floats |
| %l |  | long |
| %c |  | character |

for formatted output you use printf printf("control string", arg1, arg2,...argn);

/\* program to exhibit i/o \*/

#include<stdio.h>

main()

{

int a,b; float c; printf("Enter any number"); a=getchar();

printf("the char is ");

putchar(a);

printf("Exhibiting the use of scanf"); printf("Enter three numbers"); scanf("%d%d%f",&a,&b,&c); printf("%d%d%f",a,b,c);

}

**OPERATORS :**

An operator is a symbol that tells the compiler to perform certain mathematical or logical manipulations. They form expressions.

C operators can be classified as

1. Arithmetic operators
2. Relational operators
3. Logical operators
4. Assignment operators
5. Increment or Decrement operators
6. Conditional operator
7. Bit wise operators
8. Special operators
9. **ARITHMETIC OPERATORS :** All basic arithmetic operators are present in C.

operator meaning

+ add

- subtract

\* multiplication

/ division

% modulo division(remainder)

An arithmetic operation involving only real operands(or integer operands) is called real arithmetic(or integer arithmetic). If a combination of arithmetic and real is called mixed mode arithmetic.

1. **RELATIONAL OPERATORS :** We often compare two quantities and depending on their relation take certain decisions for that comparison we use relational operators.

|  |  |  |
| --- | --- | --- |
| operator | | meaning |
|  |  |
| < |  | is less than |
| > |  | is greater than |
| <= |  | is less than or equal to |
| >= |  | is greater than or equal to |
| == |  | is equal to |
| != |  | is not equal to |

It is the form of

ae-1 relational operator ae-2

1. **LOGICAL OPERATORS :** An expression of this kind which combines two or more relational expressions is termed as a logical expressions or a compound relational expression. The operators and truth values are

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| op-1 | op-2 | op-1 && op-2 op-1 || op-2 | | |
| non-zero | non-zero | 1 |  | 1 |
| non-zero | 0 | 0 |  | 1 |
| 0 | non-zero | 0 |  | 1 |
| 0 | 0 | 0 |  | 0 |
| op-1 | !op-1 |  |  |  |
| non-zero | zero |  |  |  |
| zero | non-zero |  |  |  |

1. **ASSIGNMENT OPERATORS :** They are used to assign the result of an expression to a variable. The assignment operator is '='.

v op=exp

v is variable op binary operator exp expression op= short hand assignment operator short hand assignment operators use of simple assignment operators use of short hand assignment operators

a=a+1 a+=1 a=a-1 a-=1 a=a%b a%=b

1. **INCREMENT AND DECREMENT OPERATORS :**

++ and == are called increment and decrement operators used to add or subtract.

Both are unary and as follows

++m or m++

--m or m--

The difference between ++m and m++ is if m=5; y=++m then it is equal to m=5;m++;y=m; if m=5; y=m++ then it is equal to m=5;y=m;m++;

1. **CONDITIONAL OPERATOR :** A ternary operator pair "?:" is available in C to construct conditional expressions of the form

exp1 ? exp2 : exp3;

It work as if exp1 is true then exp2 else exp3

1. **BIT WISE OPERATORS :** C supports special operators known as bit wise operators for manipulation of data at bit level. They are not applied to float or double.

operator meaning

& Bitwise AND

|  |  |  |
| --- | --- | --- |
| | |  | Bitwise OR |
| ^ |  | Bitwise exclusive OR |
| << |  | left shift |
| >> |  | right shift |
| ~ |  | one's complement |

1. **SPECIAL OPERATORS :** These operators which do not fit in any of the above classification are ,(comma), sizeof, Pointer operators(& and \*) and member selection operators (. and ->). The comma operator is used to link related expressions together. sizeof operator is used to know the sizeof operand.

# UNIT - II

**SELECTION STATEMENTS(DECISION MAKING):**

**IF AND SWITCH STATEMENTS :**

We have a number of situations where we may have to change the order of execution of statements based on certain conditions or repeat a group of statements until certain specified conditions are met.

The if statement is a two way decision statement and is used in conjunction with an expression. It takes the following form

If(test expression)

If the test expression is true then the statement block after if is executed otherwise it is not executed

if (test expression)

{

statement block;

}

statement–x ; only statement–x is executed.

/\* program for if \*/ #include<stdio.h> main()

{

int a,b; printf(“Enter two numbers”); scanf(“%d%d”,&a,&b):

if a>b

printf(“ a is greater”); if b>a printf(“b is greater”);

}

**The if –else statement:**

If your have another set of statement to be executed if condition is false then if-else is used if (test expression)

{

statement block1;

}

else

{

statement block2;

}

statement –x ;

/\* program for if-else \*/ #include<stdio.h> main()

{

int a,b;

printf(“Enter two numbers”); scanf(“%d%d”,&a,&b):

if a>b

printf(“ a is greater”)

else

printf(“b is greater”);

}

**Nesting of if..else statement :**

If more than one if else statement

if(text cond1)

{

if (test expression2

{

statement block1;

}

else

{

statement block 2;

}

}

else

{

statement block2;

}

statement-x ;

if else ladder

if(condition1)

statement1;

else if(condition2)

statement 2;

else if(condition3)

statement n; else

default statement.

statement-x;

The nesting of if-else depends upon the conditions with which we have to deal.

**THE SWITCH STATEMENT:**

If for suppose we have more than one valid choices to choose from then we can use switch statement in place of if statements.

switch(expression)

{.

case value-1

block-1 break;

case value-2

block-2 break;

--------

--------

default:

default block;

break;

}

statement–x

In case of

if(cond1)

{

statement-1

}

if (cond2)

{

statement 2

}

/\* program to implement switch \*/ #include<stdio.h>

main()

{

int marks,index;

char grade[10]; printf(“Enter your marks”); scanf(“%d”,&marks); index=marks/10; switch(index)

{

case 10 : case 9: case 8: case 7:

case 6:

grade=”first”;

break;

case 5 :

grade=”second”;

break;

case 4 :

grade=”third”;

break;

default :

grade =”fail”;

break;

}

printf(“%s”,grade);

}

**LOOPING :**

Some times we require a set of statements to be executed repeatedly until a condition is met.

We have two types of looping structures. One in which condition is tested before entering the statement block called entry control.

The other in which condition is checked at exit called exit controlled loop.

**WHILE STATEMENT :**

While(test condition)

{

body of the loop

}

It is an entry controlled loop. The condition is evaluated and if it is true then body of loop is executed. After execution of body the condition is once again evaluated and if is true body is executed once again. This goes on until test condition becomes false.

/\* program for while \*/

#include<stdio.h> main()

{

int count,n;

float x,y;

printf(“Enter the values of x andn”);

scanf(“%f%d”,&x,&n);

y=1.0; count=1; while(count<=n)

{ y=y\*x;

count++;

}

printf(“x=%f; n=%d; x to power n = %f”,x,n,y);

}

**DO WHILE STATEMENT :**

The while loop does not allow body to be executed if test condition is false. The do while is an exit controlled loop and its body is executed at least once.

do { body

}while(test condition);

/\* printing multiplication table \*/

#include<stdio.h>

#define COL 10 #define ROW 12 main()

{

int row,col,y;

row=1;

do {

col=1;

do

{

y=row\*col;

printf(“%d”,y);

col=col+1;

}while(col<=COL); printf(“\n”);

row=row+1;

}while(row<=ROW);

}

**THE FOR LOOP :**

It is also an entry control loop that provides a more concise structure

for(initialization; test control; increment)

{

body of loop

}

/\* program of for loop \*/

#include<stdio.h> main()

{

long int p; int n;

double q;

printf(“2 to power n “);

p=1;

for(n=0;n<21;++n)

{

if(n==0)

p=1;

else p=p\*2;

q=1.0/(double)p; printf(“%101d%10d”,p,n);

}

}

**UNCONDITIONAL STATEMENTS:**

**BREAK STATEMENT:**

This is a simple statement. It only makes sense if it occurs in the body of a switch, do, while or for statement. When it is executed the control of flow jumps to the statement immediately following the body of the statement containing the break. Its use is widespread in switch statements, where it is more or less essential to get the control .

The use of the break within loops is of dubious legitimacy. It has its moments, but is really only justifiable when exceptional circumstances have happened and the loop has to be abandoned. It would be nice if more than one loop could be abandoned with a single break but that isn't how it works. Here is an example.

#include <stdio.h> #include <stdlib.h> main(){

int i;

for(i = 0; i < 10000; i++){ if(getchar() == 's') break;

printf("%d\n", i);

}

exit(EXIT\_SUCCESS); }

It reads a single character from the program's input before printing the next in a sequence of numbers. If an „s‟ is typed, the break causes an exit from the loop.

If you want to exit from more than one level of loop, the break is the wrong thing to use.

**CONTINUE STATEMENT:**

This statement has only a limited number of uses. The rules for its use are the same as for break, with the exception that it doesn't apply to switch statements. Executing a continue starts the next iteration of the smallest enclosing do, while or for statement immediately. The use of continue is largely restricted to the top of loops, where a decision has to be made whether or not to execute the rest of the body of the loop. In this example it ensures that division by zero (which gives undefined behaviour) doesn't happen.

#include <stdio.h> #include <stdlib.h> main(){

int i;

for(i = -10; i < 10; i++){

if(i == 0) continue; printf("%f\n", 15.0/i); /\*

\* Lots of other statements .....

\*/

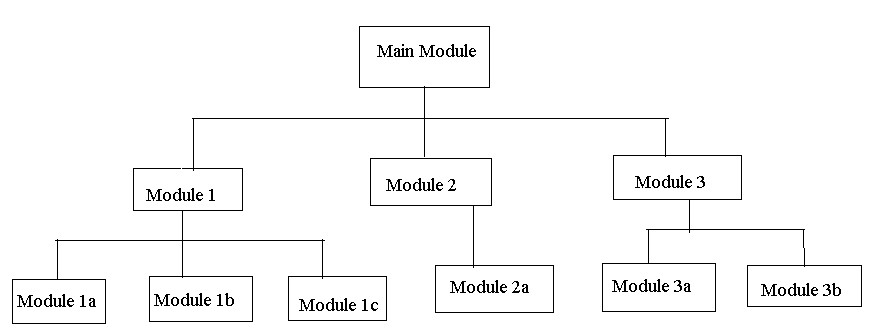
}

exit(EXIT\_SUCCESS);

# UNIT – III

**FUNCTIONS-DESIGNING STRUCTURED PROGRAMS:**

Top-down design is usually done using a visual representation of the modules known as a structure chart. The structure chart shows the relation between each module and its submodules. The structure chart is read top-down, left-right. First we read Main Module. Main Module represents our entire set of code to solve the problem.



**Structure Chart**

**FUNCTIONS :**

A function is a self contained program segment that carries out some specific well defined tasks.

**Advantages of functions:**

1. Write your code as collections of small functions to make your program modular

2.Structured programming

3.Code easier to debug

4.Easier modification

5.Reusable in other programs

**Function Definition :**

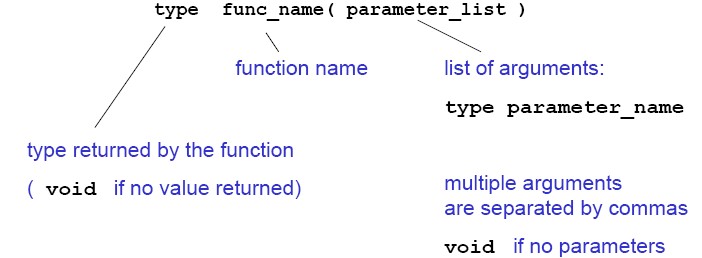
type func\_name( parameter list )

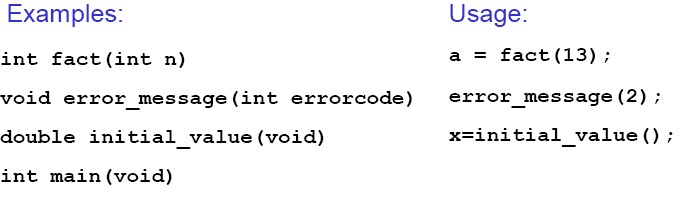
{

declarations; statements;

}

**FUNCTION HEADER :**





**TYPES OF FUNCTIONS:**

**USER DEFINED FUNCTIONS:**

Every program must have a main function to indicate where the program has to begin its execution. While it is possible to code any program utilizing only main function, it leads to a number of problems. The program may become too large and complex and as a result task of debugging, testing and maintaining becomes difficult. If a program is divided into functional parts, then each part may be independently coded and later combined into a single unit, these subprograms called “functions” are much easier to understand debug and test.

The advantages of using functions are

1. It facilitates top-down modular programming.
2. The length of a source program can be reduced by using functions at appropriate places.
3. It is easy to locate and isolate a faculty function for further investigations.
4. A function may be used by many other programs.

The form of the C functions.

function-name ( argument list ) argument declaration;

{

local variable declarations;

executable statement 1;

executabie statement 2;

* - - - - - - - - -
* - - - - - - - - -
* - - - - - - - - -

return (expression) ;

}

The return statement is the mechanism for returning a value to the calling function. All functions by default returns int type data. we can force a function to return a particular type of data by using a type specifier in the header.

A function can be called by simply using the function name in the statement.

**STANDARD LIBRARY FUNCTIONS AND HEADER FILES:**

The C language is accompanied by a number of library functions that perform various tasks.

Some of the Header files are:

<ctype.h> character testing and conversion functions.

<math.h> Mathematical functions

<stdio.h> standard I/O library functions

<stdlib.h> Utility functions such as string conversion routines memory allocation routines , random number generator,etc.

<string.h> string Manipulation functions

<time.h> Time Manipulation functions

**STDIO.H**

**Printf:** Print the text which user wants to output Screen.

**Scanf:** Through Keyboard by user

**STDLIB.H**

**Atoi:** Convert a string to an int

**Exit:** Terminate a program, return an integer value **free:** Release allocated memory **malloc:** Allocate memory **rand:** Generate a pseudo-random number **system:** Execute an external command

**The following table contains some more header files:**

|  |  |
| --- | --- |
| [**<assert.h>**](http://en.wikipedia.org/wiki/Assert.h) | Contains the [assert](http://en.wikipedia.org/wiki/Assertion_%28computing%29) macro, used to assist with detecting logical errors and other types of bug in debugging versions of a program. |
| [**<complex.h>**](http://en.wikipedia.org/wiki/Complex.h) | A set of functions for manipulating [complex numbers.](http://en.wikipedia.org/wiki/Complex_number) (New with **C99**) |
| [**<ctype.h>**](http://en.wikipedia.org/wiki/Ctype.h) | Contains functions used to classify characters by their types or to convert between upper and lower case in a way that is independent of the used [character set](http://en.wikipedia.org/wiki/Character_set) (typically [ASCII](http://en.wikipedia.org/wiki/ASCII) or one of its extensions, although implementations utilizing [EBCDIC](http://en.wikipedia.org/wiki/EBCDIC) are also known). |
| [**<errno.h>**](http://en.wikipedia.org/wiki/Errno.h) | For testing error codes reported by library functions. |
| [**<fenv.h>**](http://en.wikipedia.org/w/index.php?title=Fenv.h&action=edit&redlink=1) | For controlling [floating-point](http://en.wikipedia.org/wiki/Floating-point) environment. (New with **C99**) |
| [**<float.h>**](http://en.wikipedia.org/wiki/Float.h) | Contains defined constants specifying the implementation-specific properties of the [floating-point](http://en.wikipedia.org/wiki/Floating-point) library, such as the minimum difference between two different floating-point numbers (\_EPSILON), the maximum number of digits of accuracy (\_DIG) and the range of numbers which can be represented (\_MIN, \_MAX). |
| [**<inttypes.h>**](http://en.wikipedia.org/wiki/Inttypes.h) | For precise conversion between integer types. (New with **C99**) |

|  |  |
| --- | --- |
| [**<iso646.h>**](http://en.wikipedia.org/wiki/Iso646.h) | For programming in [ISO 646](http://en.wikipedia.org/wiki/ISO_646) variant character sets. (New with **NA1**) |
| [**<limits.h>**](http://en.wikipedia.org/wiki/Limits.h) | Contains defined constants specifying the implementation-specific properties of the integer types, such as the range of numbers which can be represented (\_MIN, \_MAX). |
| [**<locale.h>**](http://en.wikipedia.org/wiki/Locale.h) | For setlocale() and related constants. This is used to choose an appropriate [locale.](http://en.wikipedia.org/wiki/Internationalization_and_localization) |
| [**<math.h>**](http://en.wikipedia.org/wiki/Math.h) | For computing common mathematical functions |
| [**<setjmp.h>**](http://en.wikipedia.org/wiki/Setjmp.h) | Declares the macros setjmp and longjmp, which are used for non-local exits |
| [**<signal.h>**](http://en.wikipedia.org/wiki/Signal.h) | For controlling various exceptional conditions |
| [**<stdarg.h>**](http://en.wikipedia.org/wiki/Stdarg.h) | For accessing a varying number of arguments passed to functions. |
| [**<stdbool.h>**](http://en.wikipedia.org/wiki/Stdbool.h) | For a boolean data type. (New with **C99**) |
| [**<stdint.h>**](http://en.wikipedia.org/wiki/Stdint.h) | For defining various integer types. (New with **C99**) |
| [**<stddef.h>**](http://en.wikipedia.org/wiki/Stddef.h) | For defining several useful types and macros. |
| [**<stdio.h>**](http://en.wikipedia.org/wiki/Stdio.h) | Provides the core input and output capabilities of the C language. This file includes the venerable [printf](http://en.wikipedia.org/wiki/Printf) function. |
| [**<stdlib.h>**](http://en.wikipedia.org/wiki/Stdlib.h) | For performing a variety of operations, including conversion, [pseudo-random numbers,](http://en.wikipedia.org/wiki/Pseudorandom_number_generator) memory allocation, process control, environment, signalling, searching, |
|  | and sorting. |
| [**<string.h>**](http://en.wikipedia.org/wiki/String.h) | For manipulating several kinds of strings. |
| [**<tgmath.h>**](http://en.wikipedia.org/w/index.php?title=Tgmath.h&action=edit&redlink=1) | For type-generic mathematical functions. (New with **C99**) |
| [**<time.h>**](http://en.wikipedia.org/wiki/Time.h) | For converting between various time and date formats. |
| [**<wchar.h>**](http://en.wikipedia.org/w/index.php?title=Wchar.h&action=edit&redlink=1) | For manipulating wide streams and several kinds of strings using wide characters  - key to supporting a range of languages. (New with **NA1**) |
| [**<wctype.h>**](http://en.wikipedia.org/w/index.php?title=Wctype.h&action=edit&redlink=1) | For classifying wide characters. (New with **NA1**) |

**CATEGORIES OF FUNCTIONS :**

A function depending on whether arguments are present or not and whether a value is returned or not may belong to.

1. Functions with no arguments and no return values.
2. Functions with arguments and no return values.
3. Functions with arguments and return values.

**1. Functions with no arguments and no return values :**

When a function has no arguments, it does not receive any data from calling function. In effect, there is no data transfer between calling function and called function.

#include<stdio.h> main()

{

printline();

value(); printline();

}

printline()

{

int i;

for(i=1;i<=35;i++0) printf(“%c”,”-“);

printf(“\n”);

}

value()

{

int year, period; float inrate,sum,principal;

printf(“Enter Principal, Rate,Period”); scanf(“%f%f%f”,&principal,&inrate,&period); sum=principal;

year=1;

while(year<=period)

{

sum=sum\*(1+inrate); year=year+1;

}

printf(“%f%f%d%f”,principal,inrate,period,sum);

}

**2. Arguments but no return values :**

The function takes argument but does not return value.

The actual (sent through main) and formal(declared in header section) should match in number, type and order.

In case actual arguments are more than formal arguments, extra actual arguments are discarded. On other hand unmatched formal arguments are initialized to some garbage values.

#include<stdio.h> main()

{

float prin,inrate; int period;

printf(“Enter principal amount, interest”);

printf(“rate and period\n”); scanf(“%f%f%d”,&principal,&inrate,&period); printline(‘z’);

value(principal, inrate, peiod); printline(‘c’);

}

printline(ch) char ch;

{

int i;

for(i=1;i<=52;i++)

printf(“%c”,ch); printf(“\n”);

}

**PARAMETER PASSING TECHNIQUES:**

Parameter passing mechanism in „C‟ is of two types.

1. Call by Value
2. Call by Reference.

The process of passing the actual value of variables is known as Call by Value.

The process of calling a function using pointers to pass the addresses of variables is known as Call by Reference. The function which is called by reference can change the value of the variable used in the call.

**Example of Call by Value:**

#include <stdio.h>

void swap(int,int);

main()

{

int a,b;

printf(“Enter the Values of a and b:”); scanf(“%d%d”,&a,&b); printf(“Before Swapping \n”);

printf(“a = %d \t b = %d”, a,b);

swap(a,b);

printf(“After Swapping \n”);

printf(“a = %d \t b = %d”, a,b);

}

void swap(int a, int b)

{

int temp; temp = a; a = b; b = temp;

}

**Example of Call by Reference:**

#include<stdio.h> main()

{

int a,b; a = 10; b = 20;

swap (&a, &b);

printf(“After Swapping \n”);

printf(“a = %d \t b = %d”, a,b);

}

void swap(int \*x, int \*y)

{

int temp; temp = \*x;

\*x = \*y;

\*y = temp;

}

**STORAGE CLASSES :**

In ‘C’ a variable can have any one of four Storage Classes.

1. Automatic Variables
2. External Variables 3. Static Variables

4. Register Variables

**SCOPE :**

The Scope of variable determines over what parts of the program a variable is actually available for use.

**AUTOMATIC VARIABLES :**

They are declared inside a function in which they are to be utilized. They are created when function is called and destroyed automatically when the function is exited, hence the name automatic. Automatic Variables are therefore private (or local) to the function in which they are declared. Because of this property, automatic variables are also referred to as local or internal variables.

By default declaration is automatic. One important feature of automatic variables is that their value cannot be changed accidentally by what happens in some other function in the program. #include<stdio.h>

main()

{

int m=1000;

func2(); printf(“%d\n”,m);

}

func1()

{

int m=10;

printf(“%d\n”,m);

}

func2()

{

int m=100;

func1(); printf(“%d”,m);

}

First, any variable local to main will normally live throughout the whole program, although it is active only in main.

Secondly, during recursion, nested variables are unique auto variables, a situation similar to function nested auto variables with identical names.

**EXTERNAL VARIABLES :**

Variables that are both alive and active throughout entire program are known as external variables. They are also known as Global Variables. In case a local and global have same name local variable will have precedence over global one in function where it is declared.

#include<stdio.h>

int x;

main()

{

x=10;

printf(“%d”,x);

printf(“x=%d”,fun1());

printf(“x=%d”,fun2());

printf(“x=%d”,fun3());

}

fun1()

{

x=x+10;

return(x);

}

fun2()

{

int x;

x=1;

return(x);

}

fun3()

{

x=x+10;

return(x);

}

An externwithin a function provides the type information to just that one function.

**STATIC VARIABLES :**

The value of Static Variable persists until the end of program. A variable can be declared Static using Keyword Static like Internal & External Static Variables are differentiated depending whether they are declared inside or outside of auto variables, except that they remain alive throughout the remainder of program.

#include<stdio.h> main()

{

int I;

for (I=1;I<=3;I++)

stat();

}

stat()

{

static int x=0;

x=x+1;

printf(“x=%d\n”,x);

}

**REGISTER VARIABLES :**

We can tell the Compiler that a variable should be kept in one of the machines registers, instead of keeping in the memory. Since a register access is much faster than a memory access, keeping frequently accessed variables in register will lead to faster execution

Syntax:

register int Count.

**RECURSIVE FUNCTIONS:**

Recursion is a repetitive process in which a function calls itself (or) A function is called recursive if it calls itself either directly or indirectly. In C, all functions can be used recursively.

**Example: Fibonacci Numbers**

A recursive function for Fibonacci numbers (0,1,1,2,3,5,8,13...)

**/\* Function with recursion\*/** int fibonacci(int n)

{

if (n <= 1) return n; else return (fibonacci(n-1) + fibonacci(n-2));

}

With recursion 1.4 x 10 9 function calls needed to find the 43rd Fibonacci number(which has the value 433494437) .If possible, it is better to write iterative functions.

int factorial (int n) **/\* iterative version \*/**

{ for ( ; n > 1; --n) product \*= n; return product;

}

# UNIT – IV

**ARRAYS :**

An array is a group of related data items that share a common name.

Ex:- Students

The complete set of students are represented using an array name students. A particular value is indicated by writing a number called index number or subscript in brackets after array name. The complete set of value is referred to as an array, the individual values are called elements.

**ONE – DIMENSIONAL ARRAYS :**

A list of items can be given one variable index is called single subscripted variable or a one-dimensional array.

The subscript value starts from 0. If we want 5 elements the declaration will be

int number[5];

The elements will be number[0], number[1], number[2], number[3], number[4] There will not be number[5]

**Declaration of One - Dimensional Arrays :**

Type variable – name [sizes];

Type – data type of all elements Ex: int, float etc.,

Variable – name – is an identifier

Size – is the maximum no of elements that can be stored.

Ex:- float avg[50]

This array is of type float. Its name is avg. and it can contains 50 elements only. The range starting from 0 – 49 elements.

**Initialization of Arrays :**

Initialization of elements of arrays can be done in same way as ordinary variables are done when they are declared.

Type array name[size] = {List of Value};

Ex:- int number[3]={0,0,0};

If the number of values in the list is less than number of elements then only that elements will be initialized. The remaining elements will be set to zero automatically. Ex:- float total[5]= {0.0,15.75,-10};

The size may be omitted. In such cases, Compiler allocates enough space for all initialized elements.

int counter[ ]= {1,1,1,1};

/\* Program Showing one dimensional array \*/

#include<stdio.h>

main()

{ int i;

float x[10],value,total; printf(“Enter 10 real numbers\n”); for(i=0;i<10;i++)

{

scanf(“%f”,&value);

x[i]=value;

}

total=0;

for(i=0;i<10;i++)

total=total+x[i]

for(i=0;i<10;i++) printf(“x\*%2d+=%5.2f\n”,I+1,x\*I+); printf(“total=%0.2f”,total);

}

**STRINGS(CHARACTER ARRAYS) :**

A String is an array of characters. Any group of characters (except double quote sign )defined between double quotes is a constant string.

Ex: “C is a great programming language”.

If we want to include double quotes.

Ex: “\”C is great \” is norm of programmers “.

**Declaring and initializing strings :-**

A string variable is any valid C variable name and is always declared as an array.

char string name [size];

size determines number of characters in the string name. When the compiler assigns a character string to a character array, it automatically supplies a null character (‘\0’) at end of String. Therefore, size should be equal to maximum number of character in String plus one.

String can be initialized when declared as

1. char city\*10+= “NEW YORK’;
2. char city[10]= ,‘N’,’E’,’W’,’ ‘,’Y’,’O’,’R’,’K’,’/0’-; 3.

C also permits us to initializing a String without specifying size.

Ex:- char Strings\* += ,‘G’,’O’,’O’,’D’,’\0’-;

**READING STRINGS FROM USER:**

%s format with scanf can be used for reading String.

char address[15]; scanf(“%s”,address);

The problem with scanf function is that it terminates its input on first white space it finds. So scanf works fine as long as there are no spaces in between the text.

**Reading a line of text :**

If we are required to read a line of text we use getchar(). which reads a single characters. Repeatedly to read successive single characters from input and place in character array.

/\* Program to read String using scanf & getchar \*/

#include<stdio.h>

main()

{

char line[80],ano\_line[80],character;

int c;

c=0;

printf(“Enter String using scanf to read \n”); scanf(“%s”, line);

printf(“Using getchar enter new line\n”);

do

{

character = getchar(); ano\_line[c] = character;

c++;

- while(character !=’\n’);

c=c-1;

ano\_line\*c+=’\0’;

}

# UNIT – V

**POINTERS:**

One of the powerful features of C is ability to access the memory variables by their memory address. This can be done by using Pointers. The real power of C lies in the proper use of Pointers.

A pointer is a variable that can store an address of a variable (i.e., 112300).We say that a pointer points to a variable that is stored at that address. A pointer itself usually occupies 4 bytes of memory (then it can address cells from 0 to 232-1).

**Advantages of Pointers:**

1. A pointer enables us to access a variable that is defined out side the function.
2. Pointers are more efficient in handling the data tables.
3. Pointers reduce the length and complexity of a program.
4. They increase the execution speed.

**Definition :**

A variable that holds a physical memory address is called a pointer variable or Pointer.

**Declaration :**

Datatype \* Variable-name;

Eg:- int \*ad; /\* pointer to int \*/

char \*s; /\* pointer to char \*/ float \*fp; /\* pointer to float \*/ char \*\*s; /\* pointer to variable that is a pointer to char \*/

A pointer is a variable that contains an address which is a location of another variable in memory.

Consider the Statement p=&i; Here „&‟ is called address of a variable.

„p‟ contains the address of a variable i.

The operator & returns the memory address of variable on which it is operated, this is called Referencing.

The \* operator is called an indirection operator or dereferencing operator which is used to display the contents of the Pointer Variable.

Consider the following Statements :

int \*p,x;

x =5;

p= &x;

Assume that x is stored at the memory address 2000. Then the output for the following printf statements is :

|  |  |
| --- | --- |
|  | **Output** |
| Printf(“The Value of x is %d”,x); | 5 |
| Printf(“The Address of x is %u”,&x); | 2000 |
| Printf(“The Address of x is %u”,p); | 2000 |
| Printf(“The Value of x is %d”,\*p); | 5 |
| Printf(“The Value of x is %d”,\*(&x)); | 5 |

**POINTERS WITH ARRAYS :**

When an array is declared, elements of array are stored in contiguous locations. The address of the first element of an array is called its base address.

Consider the array

2000 2002 2004 2006 2008

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  |  |  |  |  |

a[0] a[1] a[2] a[3] a[4]

The name of the array is called its base address.

i.e., a and k& a[20] are equal.

Now both a and a[0] points to location 2000. If we declare p as an integer pointer, then we can make the pointer P to point to the array a by following assignment

P = a;

We can access every value of array a by moving P from one element to another.

|  |  |  |  |
| --- | --- | --- | --- |
| i.e., | P |  | points to 0th element |
|  | P+1 |  | points to 1st element |
|  | P+2 |  | points to 2nd element |
|  | P+3 |  | points to 3rd element |
|  | P +4 |  | points to 4th element |