***Overview of C***

C is a structured, high-level, machine independent language. It allows software developers to develop programs without worrying about the hardware platforms where they will be implemented.

The main is a part of every C program. C permits different forms of main statement. Following forms are allowed.

main() int main() void main() main(void) void main(void) int main(void)

The empty pair of parentheses indicates that the function has no arguments. This may be explicitly indicated by using keyword void inside parentheses. Keyword int or void are specified before the word main. void means that the function does not return any information to the operating system and int means that the function returns an integer value to the operating system.

A simple program with main(), printf() ,scanf () is as follows:

#include <stdio.h>

int main()

{ int a, b, result;

printf("Enter first number : ");

scanf("%d", &a);

printf("Enter second number : ");

scanf("%d", &b);

result = a + b;

printf("Sum : %d\n", result);

return 0;

}

***Data type,Variables &Constant***

**The Basic Data Types:**

The basic data types: character, integer, floating-point, double floating-point, and valueless. These are declared using char, int, float, double, and void, respectively.

**Modifying the Basic Types**

Except type void, the basic data types may have various modifiers preceding them. A type modifier

alters the meaning of the base type to more precisely fit a specific need. The list of modifiers is:

signed

unsigned

long

short

The int base type can be modified by signed, short, long, and unsigned. The char type can be

modified by unsigned and signed.

**Type Bits Minimal Range**

char 8 –127 to 127

unsigned char 8 0 to 255

signed char 8 –127 to 127

int 16 or 32 –32,767 to 32,767

unsigned int 16 or 32 0 to 65,535

signed int 16 or 32 –32,767 to 32,767

float 32 1E–37 to 1E+37 (6 digits of precision)

double 64 1E–37 to 1E+37 (10 digits of precision)

**Keywords & Identifier:**

In ‘C’ every word can be either a keyword or an identifier. Keywords have fixed meanings, and the meaning cannot be changed. They act as a building block of a ‘C’ program. There are a total of 32 keywords in ‘C’. Keywords are written in lowercase letters. For example:

auto long do

case void typedef

int for

**Variable:**

A variable is a container (storage area) to hold data.A single variable can be used at multiple locations in a program. A variable name must be meaningful. It should represent the purpose of the variable.

1. The name of the variable must not begin with a digit.

2. A variable name can consist of digits, alphabets, and even special symbols such as an underscore ( \_ ).

3. A variable name must not have any keywords, for instance, float, int, etc.

4. There must be no spaces or blanks in the variable name.

5. The C language treats lowercase and uppercase very differently, as it is case sensitive. Usually, we keep the name of the variable in the lower case.

**Declaring Variable**

Variables can be declared in three places: inside functions, in the definition of function parameters, and outside of all functions. These positions correspond to local variables, formal parameters, and global variables, respectively.

datatype v1, v2…….

**Local Variables**

Variables that are declared inside a function are called local variables. Local variables exist only while the block of code in which they are declared is executing. That is, a local variable is created upon entry into its block and destroyed upon exit. Furthermore, a variable declared within one code block has no bearing on or relationship to another variable with the same name declared within a different code block.

**Global Variables**

Unlike local variables, global variables are known throughout the program and may be used by any piece of code. Also, they will hold their value throughout the program's execution. Global variables are created by declaring them outside of any function. Any expression may access them, regardless of what block of code that expression is in. In most cases in C, It is declared outside main()

#include <stdio.h>

int x = 5; // global variable

int main()

{

 int y = 10; // local variable

    return 0; }

**User-Defined Data Types in C**

This basically allows a programmer to provide a definition to an identifier that will represent a data type which already exists in a program. The C program consists of the following types of UDT:

* Structures
* Union
* Typedef
* enum

**Structures**

We use the structure for organising a group of various data items into one single entity – for grouping those data items that are related but might belong to different data types.

struct sample {

int p, q;

float r, s;

char t, u;

};

struct sample vA, vB, vC; //definition of structure

**Typedef:**

It is used to define a identifier that would represent an existing data type. When using the typedef data type, the syntax would be:

typedef         existing\_data\_type         new\_type;

Let us consider the example as follows:

#include <stdio.h>

void main() {

typedef int Lessons; //statement-1

Lessons x = 17;

printf(“Given value =%d\n”, x);

}

**enum**

The enum refers to a keyword that is used for creating an enumerated data type. The enum is basically a special data type (enumerated data type) that consists of a set of various named values – known as members or elements. It is for assigning different names to the integral constants. This way, the program becomes much more readable.

The format for creating the enum type:

enum identifier (value\_a, value\_b, …. , value\_z);

The enumerated data types basically allow a user to create symbolic names of their own for a list of all the constants that are related to each other.

For instance, you can create the enumerated data type for the true and false conditions this way:

#include <stdio.h>

void main() {

enum week {MON = 1, TUE, WED, THURS, FRI, SAT, SUN};

enum week off = FRI;

printf(“Week Off = %d”, birthday);

}

Here, the field name MON will be assigned with the value 1. Thus, the next field name TUE will be assigned with the value 2 automatically, and so on.

The program mentioned above will print the following output:

Week Off = 5

**#define:**

The #define directive can use any of the basic data types present in the C standard.The #define is usually used in syntax that created a constant that is used to represent numbers, strings, or other expressions. The #define directive should not be enclosed with the semicolon(;).

#include <stdio.h>

#include <conio.h>

#define PI 3.14

int main()

{ printf("%f",PI);

}

***Operators & Expressions:***

C supports a rich set of built-in operators. An operator is a symbol that tells the computer to perform certain mathematical or logical manipulations. Operators are used in programs to manipulate data and variables.

1. Arithmetic operators(+,-,% etc) : a – b –a \* b a % b

2. Relational operators(<,<=,!= etc) : –35 >= 0 False 10 < 7+5 True

3. Logical operators(&&,||, !): a > b && x == 1True only if a > b is true and x == 10 is true

False if either of them false

4. Assignment operators: x += y+1;

This is same as the statement x = x + (y+1);

5. Increment and decrement operators(++,--): ++m; or m++; – —m; or m– —

6. Conditional operators(exp1 ? exp2 : exp3):x = (a > b) ? a : b; If true ans:a If false ans:b

7. Bitwise operators:| bitwise OR ^ bitwise exclusive OR << shift left >> shift right

8. Special operator: comma operator, sizeof operator, pointer operators (& and \*) and member selection operators (. and –>).

**Comma operator:** The comma operator can be used to link the related expressions together.

value = (x = 10, y = 5, x+y);

**sizeof operator:**

The operator is normally used to determine the lengths of arrays and structures when their sizes are not known to the programmer. It is also used to allocate memory space dynamically to variables during execution of a program.

sizeof(char) sizeof(sum)

**Precedence of Arithmetic operator:**  High priority \* / % Low priority +-

x = a – b / 3 + c \* 2 - 1;

output: x = 10.00000

**Conversion in Expression:**

**1.Implicit type conversion:** automatically converts any intermediate values to the proper type so that the expression can be evaluated without losing any value. If the operands are of different types, the ‘lower’ type is automatically converted to the ‘higher’ type before the operation proceeds.

**2.Explicit conversion or casting a value:** x = (int) 7.5 output x=7

a =int ( 21.3/(int)4.5 Evaluated as 21/4 output:a= 5

***Managing Input and Output Operator***

**Reading a character:**

Reading a single character can be done by using the function getchar. The getchar() function accepts any character keyed in. This includes RETURN and TAB.

The syntax: variable\_name = getchar( );

C supports many other similar functions(character functions)which are in file ctype.h and therefore the statement #include <ctype.h>

isalnum(c) Is c an alphanumeric character?

isalpha(c) Is c an alphabetic character?

isdigit(c) Is c a digi?

isupper(c) Is c an upper case letter?

islower(c) Is c lower case letter?

**Writing a character:**

function putchar() for writing characters one at a time to the terminal. It takes the form as shown below: putchar (variable\_name); where variable\_name is a type char variable containing a character. This statement displays the character contained in the variable\_name at the terminal.

#include <stdio.h>

#include <ctype.h>

main()

{

char alphabet;

printf(“Enter an alphabet”);

putchar(‘\n’); /\* move to next line \*/

alphabet = getchar();

if (islower(alphabet))

putchar(toupper(alphabet));/\* Reverse and display \*/

else

putchar(tolower(alphabet)); /\* Reverse and display \*/

}

**Formatted input:** Formatted input refers to an input data that has been arranged in a particular format. The general form of scanf is: scanf (“control string”, arg1, arg2, ...... argn);

**Reading of strings:** A scanf function can input strings : %ws or %wc. %s terminates reading at the encounter of a blank space. Some versions of scanf : %[characters] and %[^characters]. %[characters] means the character within the brackets are permissible in the input string. If the input string contains any other character, the string will be terminated. %[^characters] does exactly the reverse.

**Mixed mode data:** scanf (“%d %c %f %s”, &count, &code, &ratio, name);

**Formatted output:** The printf statement provides certain features that can be effectively exploited to control the alignment and spacing of print-outs on the terminals. The general form of printf statemen is: printf(“control string”, arg1, arg2, ....., argn);

Control string consists of three types of items:

1.Characters that will be printed on the screen as they appear.

2.Format specification defining output format

3. Escape sequence characters such as \n, \t, and \b.

***Decision Making and Branching***

**if statement:**

The if statement may be implemented in different forms depending on the complexity of conditions to be tested. The different forms are: 1. Simple if statement 2. if.....else statement 3. Nested if....else statement 4. else if ladder. The general form of a simple if statement is

if (test expression)

{

statement-block;

}

statement-x;

The ‘statement-block’ may be a single statement or a group of statements. If the test expression is true, the statement-block will be executed; otherwise the statement-block will be skipped and the execution will jump to the statement-x.

**if...else statement:** The general form is

If (test expression)

{

True-block statement(s)

}

else

{

False-block statement(s)

}

statement-x

If the test expression is true, then the true-block statement(s), immediately following the if statements are executed; otherwise, the false-block statement(s) are executed. In either case, either true-block or false-block will be executed, not both.

**Nested if-else statement:**

if (condition1){

   if (condition2)

      stmt1;

   else

      stmt2;

}

else{

   if (condition3)

      stmt3;

   else

      stmt4;

}

**else if ladder:**

if (marks > 79)

grade = “Honours”;

else if (marks > 59)

grade = “First Division”;

else if (marks > 49)

grade = “Second Division”;

else if (marks > 39)

grade = “Third Division”;

else

grade = “Fail”;

printf (“%s\n”, grade);

**The switch statement:**

The switch statement tests the value of a given variable (or expression) against a list of values and when a match is found, a block of statements associated with that is executed.

switch (expression)

{

case value-1:

block-1

break;

case value-2:

block-2

break;

default:

default-block

break;

}

statement-x;

**The goto statement:**

The goto requires a label in order to identify the place where the branch is to be made. A label is any valid variable name, and must be followed by a colon. The label is placed immediately before the statement where the control is to be transferred.

main()

{

double x, y;

read:

scanf(“%f”, &x);

if (x < 0) goto read;

y = sqrt(x);

printf(“%f %f\n”, x, y);

goto read;

}

***Decision making and Looping***

The C language provides for three constructs for performing loop operations. They are:

1. The while statement.

2. The do statement.

3. The for statement.

**The while statement:** The while is an entry-controlled loop statement. The test-condition is evaluated and if the condition is true,then the body of loop is executed.

sum = 0;

n = 1;

while(n <= 10)

{ loop sum = sum + n \* n;

n = n+1;

}

printf(“sum = %d\n”, sum);

**The do statement:** The do of the loop, the test-condition in the while statement is evaluated. If the condition is true, the program continues to evaluate the body of the loop once again. This process continues as long as the condition is true. When the condition becomes false, the loop will be terminated and the control goes to the statement that appears immediately after the while statement.

do

printf ("Input a number\n");

loop number = getnum ( );

while (number > 0);

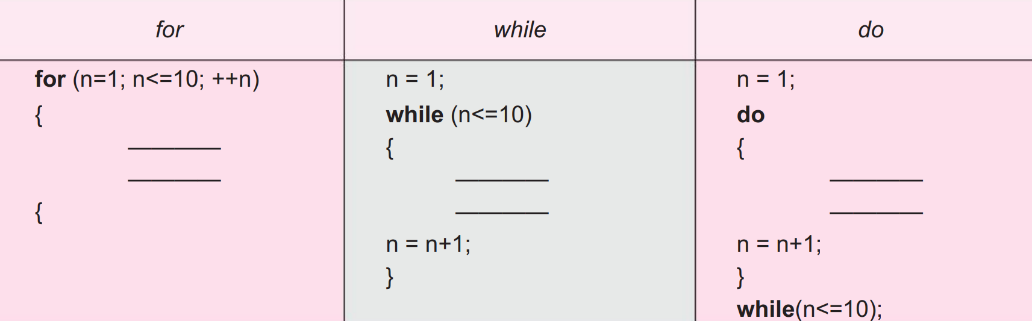
**The for loop:** The for loop is another entry-controlled loop that provides a more concise loop control structure. The general form of the for loop is :

for ( initialization ; test-condition ; increment)

{ body of the loop

}

Comparison of thee loops:



***Array***

An array is a sequenced collection of elements of the same data type. The general types of arrays.

1.One-dimensional arrays

2.Two-dimensional arrays

3.Multidimensional arrays

**The syntax of array :** datatype array-name [array size]

**Advantages of Array**

1.Arrays represent multiple data elements of the same type using a single name.

2.Accessing or searching an element in an array is easy by using the index number.

3.An array can be traversed easily just by incrementing the index by 1.

4.Arrays allocate memory in contiguous memory locations for all its data elements.

**One dimensional Array:**

The general form of array declaration is- type variable-name[ size ];

int group[10]; declares the as an array to contain a maximum of 10 integer constants. Remember:

1.Any reference to the arrays outside the declared limits would not necessarily cause an error.

Rather, it might result in unpredictable program results.

2.The size should be either a numeric constant or a symbolic constant.

**Initialization of One dimensional Array:**

**Compile time:** type array-name[size] = { list of values }; The values in the list are separated by commas. For example, int number[3] = { 0,0,0 };

Run time: An array can be explicitly initialized at run time.

#include <stdio.h>

#include <stdlib.h>

#include <ctype.h>

int main()

{ int i = 0;

char name[6];

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

{ printf("Enter the character in name array");

scanf("%c", &name[i]);

}

printf("%s", name);

return 0;

}

**Two-dimensional array:**

**Syntax:** type array\_name [row\_size] [column\_size]

int table[2][3] = { 0,0,0,1,1,1}; .The statement is equivalent as int table[2][3] = {{0,0,0}, {1,1,1}};

In the form of a matrix as shown below:

int table [2] [3] = { int table [ ] [3] = {

{ 0, 0, 0}, { 0, 0, 0},

{ 1, 1, 1} { 1, 1, 1}

}; };

Both are permitted.

**Multi-dimensional array:**

C allows arrays of three or more dimensions. The exact limit is determined by the compiler. The general form of a multi-dimensional array is type array\_name[s1][s2][s3]....[sm]; where s1 is the size of the with dimension.

int survey[3][5][12]; ]; survey is a three-dimensional array declared to contain 180 integer type elements.

float table[5][4][5][3];

**Static Arrays:**

The process of allocating memory at compile time is known as Static memory allocation and the arrays that receive static memory allocation are called static arrays.

**Dynamic array:**

Dynamic arrays are created using what are known as memory management functions malloc,calloc and realloc The concept of dynamic arrays is used in creating and manipulating data structures such as linked lists, stacks and queues.

***Character Arrays & strings***

A string is a sequence of characters that is treated as a single data item. In C, a string is a null-terminated character array. (A null is zero.) Any group of characters defined between double quotation marks is a string constant. Example:

printf(“ Well Done !”); output: Well Done !

**Syntax:** The general form of declaration of a string variable is:

char string\_name[ size ];

**Initializing Arrays:**

char city [9] = “ NEW YORK “;

char city [9]={‘N’,‘E’,‘W’,‘ ‘,‘Y’,‘O’,‘R’,‘K’,‘\0’};

char string [ ] = {‘G’,‘O’,‘O’,‘D’,‘\0’};

**Reading String from terminal:**

The input function scanf can be used with %s to read string. It terminates on the first white space

char address[10]

scanf(“%s”, address);

Convenient way of reading string with whitespace is the library function gets available in the <stdio.h>.

**Writing Strings to screen:**

The printf function with %s format to print strings to the screen. The format %s can be used to display an array of characters that is terminated by the null character.

char line [80];

gets (line);

printf (“%s”, line);

or, combining last two lines-- printf(“%s”, gets(line));

**String manipulation:**

C supports a wide range of functions that manipulate strings. The most common are listed here:

Name Function

strcpy(s1, s2) Copies s2 into s1

strcat(s1, s2) Concatenates s2 onto the end of s1

strlen(s1) Returns the length of s1

strcmp(s1,s2) Returns 0 if s1 and s2 are the same; less than 0 if s1<s2; greater than 0 if s1 >s2

strchr(s1, ch) Returns a pointer to the first occurrence of ch in s1

strstr(s1, s2) Returns a pointer to the first occurrence of s2 in s1

**Comparison of two Strings:**

C does not support comparison of 2 strings directly.

if(name1 == name2)

if(name == “ABC”)

Comparison is done character by character. until there is a mismatch or one of the strings terminates into a null character.

i =0;

while(str1[i] == str2[i] && str1[i] != ‘\0’

&& str2[i] != ‘\0’)

i = i+1;

if (str1[i] == ‘\0’ && str2[i] == ‘\0’)

printf(“strings are equal\n”);

else

printf(“strings are not equal\n”);

***User Defined Function***

**A user-defined function:**

A user-defined function (UDF) is a function provided by the user of a program or environment, in a context where the usual assumption is that functions are built into the program or environment. UDFs are usually written for the requirement of its creator.

**A built-in function:**

A built-in function is a function that is already available in a programming language, application, or another tool that can be accessed by end users.

The standard library functions are built-in functions in C programming. These functions are defined in header files. For example, The printf() is a standard library function to send formatted output to the screen (display output on the screen). This function is defined in the <stdio.h> header file.

**Advantage of user defined function:**

1. **Reduction in Program Size:**Since any sequence of statements which are repeatedly used in a program can be combined together to form a user defined functions. And this functions can be called as many times as required. This avoids writing of same code again and again reducing program size.
2. **Reducing Complexity of Program:**Complex program can be decomposed into small sub-programs or user defined functions.
3. **Easy to Debug and Maintain :** During debugging it is very easy to locate and isolate faulty functions. It is also easy to maintain program that uses user defined functions.
4. **Readability of Program:** Since while using user defined function, a complex problem is divided in to different sub-programs with clear objective and interface which makes easy to understand the logic behind the program.
5. **Code Reusability:** Once user defined function is implemented it can be called or used as many times as required which reduces code repeatability and increases code reusability.

Control of the program is transferred to the user-defined function by calling it.

**Syntax of function call:**

functionName(argument1, argument2, ...);

**Recursion:**

Recursion is the process of repeating items in a self-similar way. In programming languages, if a program allows to call a function inside the same function, then it is called a recursive call of the function.

factorial(int n)

{ int fact;

if (n==1)

return(1);

else

fact = n\*factorial(n-1);

return(fact);

}

***Structure and Union***

Structure is a user-defined datatype in C language which allows us to combine data of different types together. Structure helps to construct a complex data type which is more meaningful. It is somewhat similar to an Array, but an array holds data of similar type only.

struct keyword is used to define a structure. struct defines a new data type which is a collection of primary and derived data types.

**The general format: For Example:**

struct tag\_name struct book\_bank

{ {

data\_type member1; char title[20];

data\_type member2; char author[15];

-------------------- int pages;

----------------- float price;

}; };

**Type Defined Structure:**

The keyword typedef is used to define a structure.

typedef struct

{ . . . . .

type member1;

type member2;

. . . . .

} type\_name;

**Copying & Comparing Structure variables:**

Two variables of same structure type can be copied like normal variables.

person1 = person2;

person2 = person1;

C does not permit logical operations on structure variables.

person1 == person2

person1 != person2 are invalid.

**Operations on individual member:**

A member with dot operator with structure variable is treated as a normal variable.

if (student1.number == 111)

student1.marks += 10.00;

float sum = student1.marks + student2.marks;

student2.marks \* = 0.5;

**Arrays within Structure:**

C permits use of single dimension or multi dimension array of int or float type as structure member.

struct marks

{ int number;

float subject[3];

} student[2];

**Structure Within Structure:**

struct salary

{ char name;

char department;

struct

{ int dearness;

int house\_rent;

int city;

}

allowance;

}

employee;

Structure and Function:

General format is- function\_name(structure\_variable\_name);

**Union:**

Unions may contain many members of but can only handle one member at a time.

union item

{

int m;

float x;

char c;

} code;

They can be initialized at the time of declaration.

***Pointers***

A pointer is a derived data type in C. Pointers contain memory addresses as their values. Since these memory addresses are the locations in the computer memory where program instructions and data are stored, pointers can be used to access and manipulate data stored in the memory.

**Declaring pointer variables:**

data\_type \*pt\_name;

It tells the compiler that:

1.The asteric \* tells that the pt\_name is a pointer variable.

2. pt\_name needs a memory location.

3. pt\_name points to a variable of type data\_type.

**Pointer declaration:**

int\* p Or, int \*p Or, int \* p

**Pointer Initialization:**

int quantity;

int \*p; /\* declaration \*/

p = &quantity; /\* initialization \*/

It can also be initialized with declaration : int \*p = &quantity;

**Accessing a variable through pointer:**

int quantity, \*p, n;

quantity = 179;

p = &quantity;

n = \*p;

Pointer Expressions:

pointer variables can be used in expressions. For example, if p1 and p2 are

properly declared and initialized pointers, then the following statements are valid.

y = \*p1 \* \*p2; same as (\*p1) \* (\*p2)

sum = sum + \*p1;

z = 5\* – \*p2/ \*p1; same as (5 \* (– (\*p2)))/(\*p1)

\*p2 = \*p2 + 10;

***File Management in C***

In C programming language, the programs store results, and other data of the program to a file using file handling in C. Also, we can extract/fetch data from a file to work with it in the program. To store data in secondary the things which need to be specified are-

**1.Filename:**

A string of Characters containing two parts, a primary name and an optional period

**2. Data structure:** The general format of declaring and opening a file-

FILE \*fp;

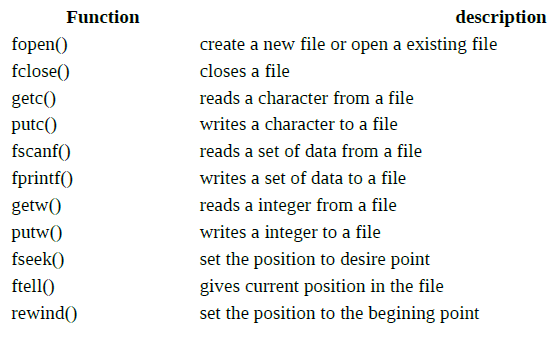
fp = fopen(“filename”, “mode”);

**3. Purpose:**

r open file for read only

w open file for write only

a open file for appending data to it.



***Dynamic Memory Allocation***

Dynamic data structure provide flexibility in adding, deleting and rearranging data items at run time. Dynamic memory management techniques permit us to allocate additional memory space or to release unwanted space at run time, thus, optimizing the use of storage space.

Function Task

malloc Allocates required size of bytes and returns a pointer to the first byte of the

allocated space.

calloc Allocates space for an array of elements, initializes them to zero and then . returns a pointer to the memory.

free Frees previously allocated space.

realloc Modifies the size of previously allocated space.

**malloc:**

The malloc function reserves a block . It takes the following form: ptr = (cast-type \*) malloc(byte-size);

ptr is a pointer of type cast-type. The malloc returns a pointer (of cast-type) to an area of memory with size byte-size.

**calloc:**

calloc allocates multiple blocks of storage, each of the same size, and then sets all bytes to zero. The general form of calloc is: ptr = (cast-type \*) calloc (n, elem-size);

**realloc:**

If the memory allocation is larger or smaller than needed e can change the memory size already allocated can be changed with the help of the function realloc. This process is called the reallocation of memory.

ptr = malloc(size);

then reallocation of space may be done by the statement

ptr = realloc(ptr, newsize);

**free:**

Data stored in block of memory can be released for future use, using the free function:

free (ptr);

ptr is a pointer to a memory block, which has already been created by malloc or calloc.