INSTITUTE FOR ADVANCED COMPUTING AND SOFTWARE DEVELOPMENT(IACSD),AKURDI

C Programming

PG-DAC SEP 2021

Table of Contents

Introduction to C **1**

Control Statements **9**

Arrays **12**

Functions **17**

Pointers **23**

Structures **28**

Preprocessing in C **33**

File Handling **36**

Chapter 1

Introduction to C

Introduction to C

C is a programming language developed at AT& T's Bell Laboratories of USA in 1972. It was designed and written by Dennis Ritchie. In the late seventies C began to replace the more familiar languages of that time like PL/I, ALGOL, etc. ANSI C standard emerged in the early 1980s; this book was split into two titles: The original was still called Programming in C, and the title that covered ANSI C was called Programming in ANSI C. This was done because it took several years for the compiler vendors to release their ANSI C compilers and for them to become ubiquitous. It was initially designed for programming UNIX operating system. Now the software tool as well as the C compiler is written in C. Major parts of popular operating systems like Windows, UNIX, Linux is still written in C. This is because even today when it comes to performance (speed of execution) nothing beats C. Moreover, if one is to extend the operating system to work with new devices one needs to write device driver programs. These programs are exclusively written in C. C seems so popular is because it is reliable, simple and easy to use. often heard today is – "C has been already superseded by languages like C++, C# and Java.

Structure of C Program

- Comments if any
- Preprocessor directive
- Global variable declaration
- main function( )
- { Local variables; Statements; }
- User defined function

Comment line:

It indicates the purpose of the program. It is represented as /*…………………………….*/

Comment line is used for increasing the readability of the program. It is useful in explaining the program and generally used for documentation.

Preprocessor Directive:

#include tells the compiler to include information about the standard input/output library. It is also used in symbolic constant such as #define PI 3.14(value). The header file contains definition &declaration of system defined function such as printf ( ), scanf ( ) etc.

Global Declaration:

This is the section where variable are declared globally so that it can be accessed by all the functions used in the program. And it is generally declared outside the function main().

main( ) function:

Every source code has one main () function from where actually program is started and it is enclosed within the pair of curly braces.

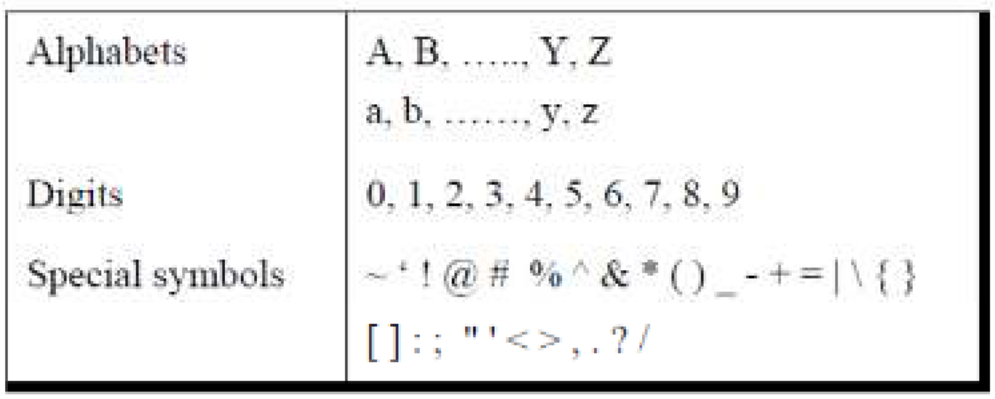
Syntax: main() { …….. …….. …….. } .
Note that at the end of each line, the semi-colon is given which indicates statement termination.

```
/*First c program*/
 #include <stdio.h>
void main ( )
{
        printf ("Welcome to IACSD.\n");
 }
```

Output: Welcome to IACSD.

Character set

A character denotes any alphabet, digit or special symbol used to represent information. Valid alphabets, numbers and special symbols allowed in C are



Alphabets	A, B, ….., Y, Z a, b, ……, y, z
Digits	0, 1, 2, 3, 4, 5, 6, 7, 8, 9
Special symbols	~ ' ! @ # % ^ & * ( ) _ - + = | \ { } [ ] : ; " ' < > , . ? /

The alphabets, numbers and special symbols when properly combined form constants, variables and keywords.

Identifiers

Identifiers are user defined word used to name of entities like variables, arrays, functions, structures etc. Rules for naming identifiers are:

1) name should only consists of alphabets (both upper and lower case), digits and underscore (_) sign.
2) first characters should be alphabet or underscore .
3) name should not be a keyword .
4) since C is a case sensitive, the upper case and lower case considered differently, for example code, Code, CODE etc. are different identifiers.
5) identifiers are generally given in some meaningful name such as value, net_salary, age, data etc. An identifier name may be long, some implementation recognizes only first eight characters, most recognize 31 characters. ANSI standard compiler recognizes 31 characters.

Keywords

There are certain words reserved for doing specific task, these words are known as reserved word or keywords. These words are predefined and always written in lower case or small letter. These keywords can't be used as a variable name as it assigned with fixed meaning. Some examples are int, short, signed, unsigned, default, volatile, float, long, double, break, continue, typedef, static, do, for, union, return, while, do, extern, register, enum, case, goto, struct, char, auto, const etc.

Data types

Data types refer to an extensive system used for declaring variables or functions of different types before its use. The type of a variable determines how much space it occupies in storage and how the bit pattern stored is interpreted. The value of a variable can be changed any time. C has the following 4 types of data types

- **Basic built-in /primitive data types**: int, float, double, char
- **Enumeration data type**: enum
- **Derived data type**: pointer, array, structure, union
- **Void data type**: void

- **Size qualifier**: short, long
- **Sign qualifier**: signed, unsigned

The type int means that the variables listed are integers; by contrast with float, which means floating point, i.e., numbers that may have a fractional part. The range of both intand float depends on the machine you are using; 16-bits ints, which lie between -32768 and+32767, are common, as are 32-bit ints. A float number is typically a 32-bit quantity, with at least six significant digits and magnitude generally between about 10^{-38} and 10^{38}.

Data Types in C

Type	Typical Size in 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	Same as int
short int	16	-32,767 to 32,767
unsigned short int	16	0 to 65,535
signed short int	16	Same as short int
long int	32	–2,147,483,647 to 2,147,483,647
long long int	64	–(263 –1) to 263 –1 (Added by C99)
signed long int	32	Same as long int
unsigned long int	32	0 to 4,294,967,295
unsigned long long int	64	264 –1 (Added by C99)
float	32	1E–37 to 1E+37 with six digits of precision
double	64	1E–37 to 1E+37 with ten digits of precision
long double	80	1E–37 to 1E+37 with ten digits of precision

Constants

Constant is a any value that cannot be changed during program execution. In C, any number, single character, or character string is known as a constant. A constant is an entity that doesn't change whereas a variable is an entity that may change.

Variables

Variable is a data name which is used to store some data value or symbolic names for storing program computations and results. The value of the variable can be change during the execution.

Operators

This is a symbol use to perform some operation on variables, operands or with the constant. Some operator required 2 operands to perform operation or some required single operation. Several operators are

1. Arithmetic Operator

This operator are used for numeric calculation. These are of either Unary arithmetic operator, Binary arithmetic operator. Where Unary arithmetic operator required only one operand such as +,-, ++, --,!, tiled. And these operators are addition, subtraction, multiplication, division. Binary arithmetic operator on other hand required two operand and its operators are +(addition), -(subtraction), *(multiplication), /(division), %(modulus).

2. Assignment Operator

A value can be stored in a variable with the use of assignment operator. The assignment operator (=) is used in assignment statement and assignment expression. Operand on the left hand side should be variable and the operand on the right hand side should be variable or constant or any expression.

3. Increment and Decrement

The Unary operator ++, --, is used as increment and decrement which acts upon single operand. Increment operator increases the value of variable by one .Similarly decrement operator decrease the value of the variable by one. It again categories into prefix post fix . In the prefix the value of the variable is incremented 1st, then the new value is used, where as in postfix the operator is written after the operand (such as m++,m--).

Similarly in the postfix increment and decrement operator is used in the operation. And then increment and decrement is perform.

4. Relational Operator

It is use to compared value of two expressions depending on their relation. Expression that contain relational operator is called relational expression. Here the value is assign according to true or false value. a.(a>=b) || (b>20) b.(b>a) && (e>b) c. 0(b!=7) 5.

5.Conditional Operator

It sometimes called as ternary operator. Since it required three expressions as operand and it is represented as (? , :).
SYNTAX exp1 ?exp2 :exp3 Here exp1 is first evaluated. It is true then value return will be exp2 .If false then exp3.
EXAMPLE

```
void main()
{int a=10, b=2,s; s=
        (a>b) ?a:b;
        printf("value is:%d");
 }
 Output: Value is:10
```

6. Comma Operator

Comma operator is use to permit different expression to be appear in a situation where only one expression would be used. All the expression are separator by comma and are evaluated from left to right. EXAMPLE int i, j, k, l;

7. Sizeof Operator

Size of operator is a Unary operator, which gives size of operand in terms of byte that occupied in the memory. It is also use to allocate size of memory dynamically during execution of the program

8. Bitwise Operator

Bitwise operator permit programmer to access and manipulate of data at bit level. Various bitwise operator enlisted are

- one's complement (~)
- bitwise AND (&)
- bitwise OR (|)
- bitwise XOR (^)
- left shift (<<)
- right shift (>>).

9. Logical or Boolean Operator

These operators are used with one or more operand and return either value zero (for false) or nonzero (for true). The operand may be constant, variables or expressions. And the expression that combines two or more expressions is termed as logical expression.

- && AND
- || OR
- ! NOT

Where logical NOT is a unary operator and other two are binary operator. Logical AND gives result true if both the conditions are true, otherwise result is false. And logical OR gives result false if both the condition false, otherwise result is true.

Precedence and Associativity of C Operators

Symbol	Type of Operation	Associativity
[ ] ( ) . –> postfix ++ and postfix ––	Expression	Left to right
prefix ++ and prefix –– sizeof& * + – ~ !	Unary	Right to left
Typecasts	Unary	Right to left
* / %	Multiplicative	Left to right
+ –	Additive	Left to right
<<>>	Bitwise shift	Left to right
<><= >=	Relational	Left to right
== !=	Equality	Left to right
&	Bitwise-AND	Left to right
^	Bitwise-exclusive-OR	Left to right
|	Bitwise-inclusive-OR	Left to right
&&	Logical-AND	Left to right
||	Logical-OR	Left to right
? :	Conditional-expression	Right to left
= *= /= %= += –= <<= >>=&= ^= |=	Simple and compound assignment	Right to left
,	Sequential evaluation	Left to right

Chapter 2

Control Statements

Control Statement

Generally C program statement is executed in a order in which they appear in the program. But sometimes we use decision making condition for execution of only a part of program, that is called control statement. Control statement defines how the control is transferred from one part to the other part of the program. There are several control statement like if...else, switch, while, do....while, for loop, break, continue, goto etc.

if…..else ... Statement

it is conditional control statement that contains one condition. Condition may be true or false, where non-zero value regarded as true& zero value regarded as false. If condition is satisfy true, then a single or block of statement executed otherwise, if condition is false then single or block of statement in else is executed. Its syntax is:-

```
if (condition)
{
Statement1;
Statement2;
}
else
{
Statement1;
Statement2;
}
```

Loops in C

Loop:-it is a block of statement that performs set of instructions. In loops repeating particular portion of the program either a specified number of time oruntil a particular no of condition is being satisfied.
There are three types of loops in c
1.while loop
2.do while loop
3.for loop

while loop
Syntax:-

```
while(condition)
{
Statement 1;
Statement 2;
}
```

Or

```
while(test condition)
        Statement;
```

The test condition may be any expression .when we want to do something a fixed no of times but not known about the number of iteration, in a program then while loop is used.

Here first condition is checked if, it is true body of the loop is executed else, If condition is false control will be come out of loop. It is pretested condition loop.

do while loop

Syntax:-

```
    do
    {
    Statement;
    }
    while(condition);
```

It is post tested condition loop i.e. condition is checked at end of every iteration.
Note: Do while loop used rarely when we want to execute a loop at least once.

for loop

In a program, for loop is generally used when number of iteration are known in advance.
The body of the loop can be single statement or multiple statements.
Syntax:-

```
    for(exp1; exp2; exp3)
    {
    Statement;
    }
```

Here exp1 is an initialization expression, exp2 is test expression or condition and exp3 is an update expression. Expression 1 is executed only once when loop started and used to initialize the loop variables. Condition expression generally uses relational and logical operators. And updating part executed only when after body of the loop is executed.

Nesting of loop

When a loop written inside the body of another loop then, it is known as nesting of loop.
Any type of loop can be nested in any type such as while, do while, for.

Break statement

Sometimes it becomes necessary to come out of the loop even before loop condition becomes false then break statement is used. Break statement is used inside loop and switch statements. It cause immediate exit from that loop in which it appears and it is generally written with condition.

Continue statement

Continue statement is used for continuing next iteration of loop after skipping some statement of loop. When it encountered control automatically passes through the beginning of the loop. It is usually associated with the if statement. It is useful when we want to continue the program without executing any part of the program.

The difference between break and continue is, when the break encountered, loop is terminated and it transfer control to the next statement and when continue is encountered control come back to the beginning of loop.

Chapter 3

Arrays in C

ARRAY

Array is the collection of similar data types or collection of similar entity stored in contiguous memory location. Array of character is a string. Each data item of an array is called an element. And each element is unique and located in separated memory location. Each of elements of an array share a variable but each element having different index no. known as subscript. An array can be a single dimensional or multi-dimensional and number of subscripts determines its dimension. Subscript always starts with zero.

DECLARATION OF AN ARRAY :

```
                    data_type  array_name [size];
          e.g. intarr[100]; //array only declared& will store garbage
```

The declaration of an array tells the compiler that, the data type, name of the array, size of the array and for each element it occupies memory space. Like for intdata type, it occupies 2 bytes for each element and for float it occupies 4 byte for each element etc.

INITIALIZATION OF AN ARRAY:

After declaration element of local array has garbage value. If it is global or static array then it will be automatically initialize with zero. Explicitly it can be initialize as,

```
                    Data_type array_name [size] = {value1, value2, value3…}
          e.g.inarr[5]={20,60,90, 100,120}
```

Array subscript always start from zero which is known as lower bound and upper value is known as upper bound and the last subscript value is one less than the sizeof array. Subscript can be an expression i.e. integer value. It can be any integer, integer constant, integer variable, integer expression or return value from functional call that yield integer value.
So if i & j are not variable then the valid subscript are ar [i*7],ar[i*i],ar[i++],ar[3];

ACCESSING OF ARRAY ELEMENT:

```
/*Write a program to input values into an array and display them*/
#include<stdio.h>
int main()
{
          intarr[5],i;
          for(i=0;i<5;i++)
          {
                    printf( “enter a value for arr[%d] \n”, I );
                    scanf(“%d”,&arr[i]);
          }
          printf(“the array elements are: \n”);
          for (i=0;i<5;i++)
          {
                    printf(“%d\t”,arr[i]);
          }
          return 0;
}
```

OUTPUT:

```
Enter a value for arr[0] = 12
Enter a value for arr[1] =45
Enter a value for arr[2] =59
Enter a value for arr[3] =98
Enter a value for arr[4] =21
The array elements are 12 45 59 98 21
```

Example: From the above example value stored in an array are and occupy its memory addresses 2000, 2002, 2004, 2006, 2008 respectively.
a[0]=12, a[1]=45, a[2]=59, a[3]=98, a[4]=21

ar[0]	ar[1]	ar[2]	ar[3]	ar[4]
12	45	59	98	21
2000	2002	2004	2006	2008

Two dimensional arrays

Two dimensional array is known as matrix. The array declaration in both the arrayi.e.in single dimensional array single subscript is used and in two dimensional array two subscripts are is used. Its syntax is

```
data_typearray_name[row][column];
```

Or we can say 2-d array is a collection of 1-D array placed one below the other.
Total no. of elements in 2-D array is calculated as row*column.
Example:-

```
int a[2][3];
```

Total no of elements=row*column is 2*3 =6
It means the matrix consist of 2 rows and 3 columns.
For example:-

```
20    2    7
8     3   15
```

Positions of 2-D array elements in an array are as below

a [0][0]	a [0][1]	a [0][2]	a [1][0]	a [1][1]	a [1][2]
20	2	7	8	3	15
2000	2002	2004	2006	2008	2010

Initialization of 2-d array: 2-D array can be initialized in a way similar to that of 1-D array.
For example:-

```
int mat[4][3]={11,12,13,14,15,16,17,18,19,20,21,22};
```

These values are assigned to the elements row wise.

Accessing 2-d array /processing 2-d array

For example

```
int a[2][3];
```

for reading value:-

```
        for(i=0;i<2;i++)
        {
                for(j=0;j<3;j++)
                  {
                scanf(“%d”, &a[i][j]);
                }
        }
```

For displaying value:-

```
        for(i=0;i<2;i++)
        {
                for(j=0;j<3;j++)
                   {
                        printf(‟%d″,a[i][j]);
                   }
        }
```

String Handling in C

Array of character is called a string. It is always terminated by the NULL character. String is a one dimensional array of character. We can initialize the string as

char name[]={‘j’,’o’,’h’,’n’,’\o’};

Here each character occupies 1 byte of memory and last character is always NULL character. Array elements of character array are also stored in contiguous memory allocation.

The terminating NULL is important because it is only the way that the function that work with string can know, where string end. String can also be initialized as;

char name[]=″John″;

Here the NULL character is not necessary and the compiler will assume it automatically.

String constant (string literal)

A string constant is a set of character that enclosed within the double quotes and is also called a literal. Whenever a string constant is written anywhere in a program it is stored somewhere in a memory as an array of characters terminated by a NULL character (‘\o’). The string constant itself becomes a pointer to the first character in array.

Example- char crr[20]= “Tajmahal”;

1000	1001	1002	1003	1004	1005	1006	1007	1008	1009
T	a	j		M	a	h	a	l	\0

↓

It is called base address.

String Library Functions

strlen	calculates the length of string
strcat	Appends one string at the end of another
strncat	Appends first n characters of a string at the end of another
strcpy	Copies a string into another
strncpy	Copies first n characters of one string into another
strcmp	Compares two strings
strncmp	Compares first n characters of two strings
strchr	Finds the first occurrence of a given character in a string
strrchr	Finds the last occurrence of a given character in a string
strstr	Finds the first occurrence of a given string in another string

strlen()

This function returns the length of the string. i.e. the number of characters in the string excluding the terminating NULL character.

strcmp()

This function is used to compare two strings. If the two string match, strcmp()return a value 0 otherwise it return a non-zero value. It compares the strings character by character and the comparison stops when the end of the string is reached or the corresponding characters in the two string are not same.
e.g. strcmp(s1,s2)

return a value:<0 when s1<s2
=0 when s1=s2
>0 when s1>s2

strcpy()

This function is used to copy one string to another string. The function strcpy(str1,str2) copies str2 to str1 including the NULL character. Here str2 is the source string and str1 is the destination string. The old content of the destination string str1 are lost.

strcat()

This function is used to append a copy of a string at the end of the other string. If the first string is ”CDAC” and second string is 〝IACSD〞 then after using this function the string becomes “CDACIACSD”. The NULL character from str1 is moved and str2 is added at the end of str1. The 2nd string str2 remains unaffected.

Chapter 4

Functions in C

FUNCTIONS

A function is a self-contained block of codes or sub programs with a set of statements that perform some specific task or coherent task when it is called. Any 'C' program contains at least one function i.e. main().

There are basically two types of function,

1. Library function
2. User defined function

The user defined functions defined by the user according to user requirement. System defined function can't be modified; it can only read and can be used. These functions are supplied with every C compiler. Source of these library function are pre complied and only object code get used by the user by linking to the code by linker.

Functions Involves:

1. Function Declaration
2. Function Definition
3. Function Calling

In system defined/Library function:

1. Function definition: predefined, precompiled, stored in the library
2. Function declaration: In header file with or function prototype.
3. Function call: By the programmer

For User defined function

Syntax:-

```
return_typefunction_name(type arg1,type arg2 …. type argn) /*function declaration*/

main()
{
   ------
   ------
   Function_name(arg1,arg2,arg3);/* calling function*/
}
/*function definition*/
return_typefunction_name(type arg1,type arg2 …. type argn)
{
Local variable declaration;
Statement1;
Statement2;
.
.
Statement-n;
return value;
}
```

EXAMPLE:

```
 /*Sum of first n Natural numbers*/
int sum (int);//function Declaration
void main()
{
        int num, ans;
        printf("enter limit");
        scanf("%d", &num);
        ans =sum(num); //function call
        printf("summation is = %d", ans);
}

//function definition
int sum(int num)
{
        int i,ans;
        for(i=0;i<=num;i++)
        {
        ans=ans+I;
        }
        return ans;
}
```

Actual Arguments

The arguments which are mentioned or used inside the function call is knows as actual argument.

Formal Arguments

The arguments which are mentioned in function definition are called formal arguments or dummy arguments.

Types of Functions

1. Function with Arguments returning value
2. Function without Arguments returning value
3. Function with Arguments returning no value
4. Function without Arguments returning no value

Advantage of function

- By using function large and difficult program can be divided in to sub programs and solved.
- When we want to perform some task repeatedly or some code is to be used more than once at different place in the program, then function avoids this repetition or rewritten over and over.
- Fixing errors become easy.
- Due to modular function it is easy to modify and test.

Call by value and call by reference

There are two way through which we can pass the arguments to the function such as call by value and call by reference.

1. Call by value

In the call by value, copy of the actual argument is passed to the formal argument and the operation is done on formal argument.

When the function is called by call by value method, it doesn't affect content of the actual argument. Changes made to formal argument are local to block of called function so when the
Control moves back to calling function the changes made is vanish.

Example:-

```
swap(int,int);
main()
{
        intx,y;
        printf("Enter two values:");
        scanf("%d%d",&x,&y);
        swap (x ,y);
        printf("Value of x=%d and y=%d",x ,y);
}
swap(int a, int b)
{
        inttemp;
        temp=a;
        a=b;
        b=temp;
}
```

Output: enter two values: 1223
Value of x=12 and y=23

2. Call by address

Instead of passing the value of variable, address or reference is passed and the function operates on address of the variable rather than value. Formal argument is copied to the actual argument, it means formal arguments calls the actual arguments. Example:-

```
swap(int *,int*);
void main()
{
        int a,b;
        printf("Enter two values:");
        scanf("%d%d",&a,&b);
        swap(&a,&b);
        printf("after changing  a=%d and b=%d",a,b);
}
swap(int *a, int *b)
{
        int k;
        k=*a;
        *a=*b;
        *b= k;
}
```

Output: enter two values: 12 32
After changing two value of a=32 and b=12

Storage Classes

Storage class in c language is a specifier, which tells the compiler where and how to store variables, its initial value and scope of the variables in a program. Table below shows all storage classes supported by C with its details.

Storage Type	**Auto**	**Register**	**Static**	**Extern**
Keyword	auto	register	static	Extern
Life	Local	Local	Within function block	Within file
Scope	Within function block	Within function block	Through out programme	Through out programme
Initial Value	Garbage	Garbage	Zero	Zero
Memory	On stack	CPU registers	Data Section	Data Section

Recursion

When function calls itself (inside function body) ,again and again then it is called as recursive function. In recursion calling function and called function are same. It is powerful technique of writing complicated algorithm in easiest way. According to recursion problem is defined in terms of itself.

Example :- /*calculate factorial of a no. using recursion*/

```
int fact(int);//function declaration
void main()
{
        int num, result;
        printf("enter a number");
        scanf("%d",&num);
        result=fact(num); //function call
        printf("factorial is =%d",result);
}
//function definition
fact (intnum)
{
If (num==0||num==1)
        return 1;
        else return(num*fact(num-1));}
```

Chapter 5

Pointers in C

POINTER

A pointer is a variable that store memory address or that contains address another variable where addresses are the location number always contains whole number. So, pointer contains always the whole number. It is called pointer because it points to a particular location in memory by storing address of that location. Syntax:

```
        data_type *pointer_ name;
```

Note :Here * before pointer indicate the compiler that variable declared as a pointer to specified data type.

e.g.

```
        int *ptr1; //pointer to integer type
        float *ptr2; //pointer to float type
        char *ptr3; //pointer to character type
```

When pointer declared, it contains garbage value i.e. it may point to any value in the memory. Two operators are used in the pointer i.e. address operator (&) and indirection operator or dereference operator (*).Indirection operator gives the values stored at a particular address.

Example:

```
void main()
{
        int i=100;
        int *ptr;
        ptr=&i; //ptr points to variable i of type
        integer printf(‟value of i=%d″,*ptr);
        printf(‟value of i=%d″,*(&i)); printf(‟address
        of i=%d″,&i); printf(‟address of i=%d″,ptr);
        printf(‟address of p=%u″,&ptr);

}
```

Pointer Arithmetic

Pointer arithmetic is different from ordinary arithmetic and it is perform relative to the data type(base type of a pointer).

Example:-

If integer pointer contains address of memory location 2000 on incrementing we get address of 2002 ,instead of 2001, because, size of the integer is of 2 bytes (16-bit compiler).

Note:-When we move a pointer, somewhere else in memory by incrementing or decrement or adding or subtracting integer, it is not necessary that, pointer still pointer to a variable of same data, because, memory allocation to the variable are done by the compiler.

Pointer to pointer

Address of pointer variable stored in some other variable is called pointer to pointer variable.

Syntax:-

```
        data_type **ptr_name;
```

Example :

```
void main()
{
        Int i=100;
        int *ptr=&i;
        int **pptr=&ptr;

        printf("value of i=%d",i);
        printf("value of i=%d",*ptr);
        printf("value of x=%d",*(&i));
        printf("value of x=%d",**pptr);
        printf("value of ptr=%u",&ptr);
        printf("address of ptr=%u",pptr);
        printf("address of i=%u",ptr);
        printf("address of pptr=%u",&pptr);
        printf("value of ptr=%u",ptr);
        printf("value of ptr=%u",&i);
}
```

Pointer & Arrays

In C there is a very close connection between pointers and arrays. In fact they are more or less one and the same thing! When you declare an array as:

```
        int a[10];
```

You are in fact declaring a pointer 'a' to the first element in the array. That is, a is exactly the same as &a[0]. The only difference between array and a pointer variable is that the array name is a constant pointer - you cannot change the location it points at. To be more precise, a[i] is exactly equivalent to *(a+i) i.e. the value pointed at by (a + i) . In the same way *(a+ 1) is the same as a[1] and so on.

Example :

```
void main()
{
        int arr[]={10,20,30,40,50};
        int *ptr;
        ptr=&arr[0]; //same as ptr=arr

        for(i=0;i<5;i++)
        {
                printf("%d",arr[i]);
                //alternative way of printing array elements by using pointer
                printf("%d",*ptr); ptr++;
                printf("%d",*(ptr+i));
                printf("%d",ptr[i]);
        }
}
```

Array of Pointers

We can declare an array of ints or an array of floats, similarly there can be an array of pointers. Since a pointer variable always contains an address, an array of pointers would be nothing but a collection of addresses.

Example :

```
#include <stdio.h>
 void main () {

/* names is an array of pointers storing addresses of names of students */
char *names[] = {
     “Sana kale”,
     "Esha lele",
     "Neha mane ",
     "Yash Roy" };

            for ( i = 0; i < 4; i++) {
                         printf("Names[%d] = %s\n", i, names[i] );
             }
}
```

Dynamic Memory Allocation

Dynamic memory allocation in c language allows the C programmer to allocate memory at runtime. It can be achieved by library function like,

malloc() : The malloc() function allocates single block of requested memory. It doesn't initialize memory at execution time, so it has garbage value initially.

calloc(): The calloc() function allocates multiple block of requested memory. It initializes all bytes to zero.

realloc(): If memory is not sufficient you can reallocate the memory by realloc() function.

free(): The memory must be released by calling free() function.

Example :

```
void main()
{
  int range,i,*ptr;
  printf("Enter number of elements: ");
  scanf("%d",&range);
  ptr=(int*)malloc(range*sizeof(int)); //memory allocated using malloc
  if(ptr==NULL)
    {
       printf("Error in memory allocation");
       exit(0);
    }
    printf("Enter data for elements : ");
    for(i=0;i<range;i++)
    {
        scanf("%d",(ptr+i));
    }
    for(i=0;i<range;i++)
    {
        printf("%d",*(ptr+i));
    }
   free(ptr);
}
```

Pointers to Functions

Just as we declare a pointer to an int, similarly we can declare a pointer to a function (or procedure). For example, the following declares a variable ptr whose type is a pointer to a function that takes an int as a parameter and returns void as result.

```
void (*ptr)(int);
```

That is, ptr is not itself a function, but rather is a variable that can point to a function, which actually stores the start of executable code. A function pointer points to code, not data.

Example :

```
void display(int a)
{
    printf("Value of a is %d\n", a);
}

void main()
{
    void (*fun_ptr)(int) = &display; //store address of fun in pointer
    fun_ptr(100); //call fun using pointer
 }
```

Chapter 6

Structures & Unions

Structures

A structure contains a number of data types grouped together. These data types may or may not be of the same type.

Example:

```
struct Employee
{
	int eno; //structure members
	char ename[20];
	double bsalary;
}
```

This defines a new data type called struct Employee. We can declare vars of this type as,
struct Employee e1,e2,e3;
This will allocate memory for vars e1,e2,e3 in memory , 32 bytes each.

Like primary variables and arrays, structure variables can also be initialized as,
struct Employee e1 = { 101,”Rama",50000 } ;

Memory Allocation of Structure vars

eid	ename	bsalary
101	Rama	50000
1000	1004	1025

To access structure members we can use dot(.) operator.
Example : we can access id of employee by e1.eid & name by e1.name etc…

Array of Structures

To declare array of structures we write,

struct Employee e[10];

Here err is array of size 10 storing data of 10 employees in memory in continuous memory. To access employees in err we can write e[0].eid,e[0].ename etc… for first employee & so on….

Example:

```
/* Array of structures */
struct Employee
{
	int eno;
	char ename[20];
	double bsalary;
};

void main( )
{
struct Employee e[10] ;
int i ;
for ( i = 0 ; i <10 ; i++ )
{
```

```
printf ( "\nEnter id, name & salary of employee " ) ;
scanf ( "%d %s %lf", &e[i].id, &e[i].name, &e[i].salary ) ;
}
printf("\n Emloyee Details are….\n");
for ( i = 0 ; i <10 ; i++ )
{
printf ( "\n%d %s %lf", e[i].id,e[i].name, e[i].salary ) ;
}
}
```

Nested structure

When a structure is within another structure, it is called Nested structure. A structure variable can be a member of another structure and it is represented as, Example:

```
struct date
{
int dd,mm,yy;
};
struct Employee
{
        int eno;
        char ename[20];
        double bsalary;
struct date dob;
};
```

Structure Pointers:

We can have a pointer pointing to a struct. Such pointers are known as ‘structure pointers’.
Example :

```
struct Employee
{
        int eno;
        char ename[20];
        double bsalary;
};

void main( )
{
struct Employee e1 = { 101,"Rama",50000 } ;
struct Employee *ptr ;
ptr = &e1 ; //ptr stores address of Employee var e1
printf ( "\n%d %s %lf", ptr->id, ptr->name, ptr->salary ) ; //accessing data in e1 using pointer
ptr
}
```

Note : To access data members of structure using pointers we have to use arrow (->) operator.

Passing structure elements to function

We can pass each element of the structure through function but passing individual element is difficult when number of structure element increases. To overcome this, we use to pass the whole structure through function instead of passing individual element. Example:

```
struct Employee
{
        int eno;
        char ename[20];
        double bsalary;
};

void display(struct Employee);

void main()
{
        struct Employee e1 = { 101,"Rama",50000 } ;
        display(e1);
}
display(struct Employee e)
{
printf("\n Id=%d, \n Name=%s ,\n Salary=%lf", e.id, e.name, e.bsalary);
}
```

UNION

Union is derived data type which is collection of dissimilar elements. Its same as that of structure , but instead of keyword struct the keyword union is used, the main difference between union and structure is that each member of structure occupies the separate memory, but in the union members share memory. Union is used for saving memory and concept is useful when it is not necessary to use all members of union at a time. Example :

```
union DateOfBirth
{
        int age;
        char dob[12];
};
union DateOfBirth U;
```

Note : Either date of birth or age can be input to variable U and will have memory allocation of 12 bytes only , which can hold only one value among given 2 members, i.e. age or dob;

Enumerations

Enumerations are unique types with values ranging over a set of named constants called

enumerators. Enumeration (or enum) is a user defined data type in C. It is used to assign names to integral constants..
Example :

```
enum DayOfWeek{Mon, Tue, Wed, Thur, Fri, Sat, Sun};

void main()
{
   enum  DayOfWeek  day;
   day = Tue;
   printf("%d",day);
}
```

Output : 1

Note : In above code Mon is default initialized to 0, Tue is initialized to 1 & so on…Also we can initialize any random value to defined constants in enum like Sat=10 then depending on that all next constants get value initialized like Sun will get auto initialized to 11.

Chapter 7

Preprocessors

Preprocessing

Preprocessors are a way of making text processing with your C program before they are actually compiled.

Preprocessor is a program that processes our source program before it is passed to the compiler. A preprocessor performs macro substitution, conditional compilation, and inclusion of named files. Lines beginning with # communicate with this preprocessor.

Macro Definition and Expansion

1. **#define** : Syntax is,

    # define identifier token-sequence //defines a macro

It causes the preprocessor to replace subsequent instances of the identifier with the given sequence of tokens. Macros are not type checked. EXAMPLE : #define ABSDIFF(a, b) ((a)>(b) ? (a)-(b) : (b)-(a))

This defines a macro to return the absolute value of the difference between its arguments.

2. **#undef** : Syntax is,

    #undef *identifier* //cancels a macro definition

This directive cancels a previous definition of the identifier by #define.

File Inclusion

# include <filename>

It causes the replacement of that line by the entire contents of the file filename.

For example, on encountering a #include <stdio.h> directive, it replaces the directive with the contents of the stdio.h header file.

If using angle brackets like the example above, the preprocessor is instructed to search for the include file along the development environment path for the standard includes.

# include "MyHeader.h"

If you use quotation marks (" "), the preprocessor is expected to search in some additional, usually user-defined, locations for the header file, and to fall back to the standard include paths only if it is not found in those additional locations.

Conditional Compilation

Six directives are available to control conditional compilation. They delimit blocks of program text that are compiled only if a specified condition is true

#if	#if *constant-expression*	This directive checks whether the *constant-expression* is true (nonzero)
#ifdef	#ifdef *identifier*	This directive checks whether the identifier is currently defined.
#ifndef	#ifndef *identifier*	This directive checks to see if the identifier is not currently defined.
#else	#else	This directive delimits alternative source text to be compiled if the condition tested for in the corresponding #if , #ifdef , or #ifndef directive is false
#elif	#elif *constant-expression*	The #elif directive performs a task similar

		to the combined use of the else-if statements in C.
#endif	#endif	This directive ends the scope of the #if , #ifdef , #ifndef , #else , or #elif directive.

Pragmas

# pragma token-sequence

This causes the preprocessor to perform an implementation-dependent action. An unrecognized pragma is ignored.

Predefined names

Several identifiers are predefined, and expand to produce special information. They, and also the preprocessor expansion operator defined, may not be undefined or redefined.

1. __LINE__ A decimal constant containing the current source line number.
2. __FILE__ A string literal containing the name of the file being compiled.
3. __DATE__ A string literal containing the date of compilation, in the form "Mmmm dd yyyy"
4. __TIME__ A string literal containing the time of compilation, in the form "hh:mm:ss"

Chapter 8

File Handling

File Handling

A file represents a sequence of bytes on the disk where a group of related data is stored. File is created for permanent storage of data. It is a readymade structure.
In C language, we use a structure pointer of file type to declare a file.

FILE *fp;

C provides a number of functions that helps to perform basic file operations.

Function	description
fopen()	create a new file or open a existing file
fclose()	closes a file
getc()	reads a character from a file
putc()	writes a character to a file
fscanf()	reads a set of data from a file
fprintf()	writes a set of data to a file
getw()	reads a integer from a file
putw()	writes a integer to a file
fseek()	set the position to desire point
ftell()	gives current position in the file rewind() set the position to the beginning point
fseek()	It is used to move the reading control to different positions
ftell()	It tells the byte location of current position of cursor in file pointer
rewind()	It moves the control to beginning of the file.

Opening a File or Creating a File

The fopen() function is used to create a new file or to open an existing file. Syntax is,

*fp = FILE *fopen(const char *filename, const char *mode);

Here filename is the name of the file to be opened and mode specifies the purpose of opening the file. Mode can be (r,w,a,r+,w+,a+) .*fp is the FILE pointer , which will hold the reference to the opened (or created) file.

Closing a File

The fclose() function is used to close an already opened file.
Syntax : int fclose(FILE *fp);

Input/Output operation on File

getc() and putc() are simplest functions used to read and write individual characters to a file.
Example:

```
main()
{
        FILE *fp;
        char ch;
        fp = fopen("my.txt", "w");
        printf("Enter data");
        while( (ch = getchar()) != EOF)
```

```
            {
                        putc(ch,fp);
            }
            fclose(fp);
            fp = fopen("my.txt", "r");
             while( (ch = getc()) != EOF)
            {
                        printf("%c",ch);
            }
            fclose(fp);
}
```

Similarly we can use functions mentioned in above table for performing input or output operation.