Government Post Graduate College Mansehra

Submitted By : Ayesha Sajid Roll Number : 207704

Submitted To : sir Zulqarnain

Department : Comp Science

Semester : 1st Assignment of Subject : Programming Fundamentals

Question 1

Answer

COMPUTER LANGUAGES

Definition

A computer language is method of communicating with computer

The computer language is defined as code or syntax which is used to write programs or any specific applications.

A programming language is a formal language comprising a set of instructions that produce various kinds of output

 Programming languages are used in computer programming to implement algorithms.

TYPES OF PROGRAMMING LANGUAGES

There are dozens of programming languages used in the industry today. Here is overviews of the 12 most important, relevant and in-demand of these languages below.

1. Python
2. Java
3. Ruby on rails
4. HTML
5. Java Script
6. C Language
7. C++
8. C#
9. Objective-C
10. Php
11. SQL
12. SWIFt

TYPES OF COMPUTER LANGUAGES

1. Machine Language

The machine language is sometimes referred to as machine code or object code which is set of binary digits 0 and 1.

These binary digits are understood and read by a computer system and interpret it easily.

It is considered a native language as it can be directly understood by a central processing unit (CPU).

The machine language is not so easy to understand, as the language uses the binary system in which the commands are written in 1 and 0 form which is not easy to interpret.

There is only one language which is understood by computer language which is machine language.

The operating system of the computer system is used to identify the exact machine language used for that particular system.

#### 2. Assembly Language

The assembly language is considered a low-level language for microprocessors and many other programmable devices.

The assembly language is also considered as second-generation language. The first generation language is machine language.

The assembly language is mostly famous for writing an operating system and also in writing different desktop applications.

The operations carried out by programmers using assembly language are memory management, registry access, and clock cycle operations.

The drawback of assembly language is the code cannot be reused and the language is not so easy to understand.

The assembly language is considered a group of other languages.

It is used to implements the symbolic representation of machine code which is used to program CPU architecture.

The other name of assembly language is assembly code.

For any processor, the most used programming language is assembly language.

#### 3. High-Level Language

The development of high-level language was done when the programmers face the issue in writing programs as the older language has portability issues which mean the code written in one machine cannot be transferred to other machines. Thus lead to the development of high-level language.

The high-level language is easy to understand and the code can be written easily as the programs written are user-friendly in a high-level language.

The other advantage of code written in a high-level language is the code is independent of a computer system which means the code can be transferred to other machines.

The high-level of language uses the concept of abstraction and also focus on programming language rather than focusing on computer hardware components like register utilization or memory utilization.

Hierarchy of Computer Languages

https://media.geeksforgeeks.org/wp-content/uploads/pl-1.png

Question 2

A Brief History of C Programming

**The Beginning**

The C programming language came out of Bell Labs in the early 1970s. According to the Bell Labs paper [*The Development of the C Language*](https://www.bell-labs.com/usr/dmr/www/chist.html) by Dennis Ritchie, “The C programming language was devised in the early 1970s as a system implementation language for the nascent Unix operating system. Derived from the typeless language BCPL, it evolved a type structure; created on a tiny machine as a tool to improve a meager programming environment.” Originally, Ken Thompson, a Bell Labs employee, desired to make a programming language for the new Unix platform. Thompson modified the BCPL system language and created B. However, not many utilities were ever written in B due to its slow nature and inability to take advantage of PDP-11 features in the operating system. This led to Ritchie improving on B, and thus creating C.

**Early Implementations and Language Standard**

The development of C was to become the basis for Unix. According to the Bell Labs paper, “By early 1973, the essentials of modern C were complete. The language and compiler were strong enough to permit us to rewrite the Unix kernel for the PDP-11 in C during the summer of the year.” This now meant that C was becoming a strong language that could, and would be, implemented across many systems. By the middle of the 1970s, the C-based Unix was used in many projects within the Bell System as well as “a small group of research-oriented industrial, academic, and

government organizations outside [Bell Labs]".

In 1978, Brian Kernighan and Dennis Ritchie published [*The C Programming Language*](https://www.amazon.com/Programming-Language-2nd-Brian-Kernighan/dp/0131103628), which would serve as the language reference until a formal standard was adopted. Five years later, the American National Standard Institute (ANSI) formed the committee, X3J11, to establish the formal standard of C. The C standard was ratified as ANSI X3.159-1989 “Programming Language C”. This was the first formal standard of C. Currently, we are on the fourth standard of C, known as C18 as it was published in June of 2018 [JavaTpoint](https://www.javatpoint.com/history-of-c-language).

### **Uses Today**

According to [Toptal](https://www.toptal.com/c/after-all-these-years-the-world-is-still-powered-by-c-programming), UNIX operating systems are written in C and most of Linux is also in C. Also databases such as Oracle Database, MySQL, MS SQL Server, and PostgresSQL are at least partially written in C. C is the basis of many system kernels. Other programming languages, like Python and Perl, use compilers or interpreters that are written in C.

C has changed over the years and is still a common language to use in lower level programs, like kernels. But it is also used for many applications ranging from device drivers to other programming languages’ compilers or interpreters.

A characteristics of C Programming

The language also made way for C++, Objective-C, C#, and many more C-based languages that each have their own speciality

## **1) C as a mother language**

C language is considered as the mother language of all the modern programming languages because **most of the compilers, JVMs, Kernels, etc. are written in C language**, and most of the programming languages follow C syntax, for example, C++, Java, C#, etc.

It provides the core concepts like the [array](https://www.javatpoint.com/c-array), [strings](https://www.javatpoint.com/c-strings), [functions](https://www.javatpoint.com/functions-in-c), [file handling](https://www.javatpoint.com/file-handling-in-c), etc. that are being used in many languages like [C++](https://www.javatpoint.com/cpp-tutorial), [Java](https://www.javatpoint.com/java-tutorial), [C#](https://www.javatpoint.com/c-sharp-tutorial), etc.

## **2) C as a system programming language**

A system programming language is used to create system software. C language is a system programming language because it **can be used to do low-level programming (for example driver and kernel)**. It is generally used to create hardware devices, OS, drivers, kernels, etc. For example, Linux kernel is written in C.

It can't be used for internet programming like Java, .Net, PHP, etc.

## **3) C as a procedural language**

A procedure is known as a function, method, routine, subroutine, etc. A procedural language **specifies a series of steps for the program to solve the problem**.

A procedural language breaks the program into functions, data structures, etc.

C is a procedural language. In C, variables and function prototypes must be declared before being used.

## **4) C as a structured programming language**

A structured programming language is a subset of the procedural language. **Structure means to break a program into parts or blocks** so that it may be easy to understand.

In the C language, we break the program into parts using functions. It makes the program easier to understand and modify.

## **5) C as a mid-level programming language**

C is considered as a middle-level language because it **supports the feature of both low-level and high-level languages**. C language program is converted into assembly code, it supports pointer arithmetic (low-level), but it is machine independent (a feature of high-level).

A **Low-level language** is specific to one machine, i.e., machine dependent. It is machine dependent, fast to run. But it is not easy to understand.

A **High-Level language** is not specific to one machine, i.e., machine independent. It is easy to understand.

Question 3

Difference between high level and low language level

High level Low level

|  |  |
| --- | --- |
|  | |
| It is machine friendly language. | It is a programmiing  friendly language. |
| High level language is less memory efficient. |  |
| Low level language is high memory efficient. |
|  |
| It is easy to understand. | It is tough to understand. |
| It is simple to debug. | It is complex to debug comparatively. |
| It is simple to maintain. | It is complex to maintain comparatively. |
| It is portable. | It is non-portable. |
| It can run on any platform. | It is machine-dependent. |
| It needs compiler or interpreter for translation. | It needs assembler for translation. |
|  |

|  |  |
| --- | --- |
| High-Level Languages are easy to learn and understand. | Low-Level Languages are challenging to learn and understand. |
| They are executed slower than lower level languages because they require a translator program. | They execute with high speed. |
| They allow much more abstraction. | They allow little or no abstraction. |
| They do not provide many facilities at the hardware level. | They are very close to the hardware and help to write a program at the hardware level. |
| For writing programs, hardware knowledge is not required. | For writing programs, hardware knowledge is a must. |
| The programs are easy to modify. | Modifying programs is difficult. |
| A single statement may execute several instructions. | The statements can be directly mapped to processor instructions. |
| BASIC, Perl, Pascal, COBOL, Ruby etc are examples of High-Level Languages. | Machine language and Assembly language are Low-Level Language |

Question 4

//if first is multipe of 2nd integer

#include<conio.h>

#include<stdio.h>

int main()

{

int a,b;

printf("\t Enter 2 integers\n");

scanf("\n%d \n%d",&a, &b);

if(b%a==0)

printf("\t%d is multiple of %d",a,b);

else

printf("\t%d is not multiple of %d",a,b);

getch();

return 0;

}

Question 5

//Program shows the alphabets,numbers and special symbols with their integer equivalent

#include<stdio.h>

#include<conio.h>

int main()

{

char ch='a';

printf("Lower Case alphabets with their integer equivalent ");

while((ch>='a')&& (ch<='z'))

{

printf("\n \t Integer equivalent of %c is %d \t",ch,ch);

ch++;

}

char ch1='A';

printf("\nUppere Case Alphabet");

while((ch1>='A')&& (ch1<='Z'))

{

printf("\n \t Integer equivalent of %c %d \t",ch1,ch1);

ch1++;

}

printf("\nSpecial symbols");

for(int ch2=32;((ch2>=32)&&(ch2<=47));ch2++)

{

if(ch2==32)

printf("\n \t Integer equivalent of %d is %c Blank space\t",ch2,ch2);

else

printf("\n \t Integer equivalent of %d is %c \t",ch2,ch2);

}

printf("\nNumeric value");

for(char ch3=48 ; ((ch3>=48)&&(ch3<=57));ch3++)

{

printf("\n \t Integer equivalent of %d is %c \t",ch3,ch3);

}

getch();

return 0;

}

Question 6

//A program that input 1 five digit number an seperates the number

#include<conio.h>

#include<stdio.h>

int main()

{

int num;

printf("\tEnter 5 digit number:\n\t");

scanf("%d", &num);

int num1=num%10;

printf("%d ",num1);

int num2 =(num/10)%10;

printf("%d ",num2);

int num3 =((num/10)/10)%10;

printf("%d ",num3);

int num4 =(((num/10)/10)/10)%10;

printf("%d ",num4);

int num5 =((((num/10)/10)/10)/10)%10;

printf("%d ",num5);

getch();

return 0;

}

Question 7

//A program that display square an cubes of numbers from 1 to 10

#include<conio.h>

#include<stdio.h>

int main()

{

int num=0;

printf("Square and cube of numbers from 1 to 10");

printf("\n\t Number\t Square\t Cube");

while (num<=10)

{

printf("\n\t %d \t %d \t %d",num,num\*num,num\*num\*num );

num++;

}

getch();

return 0;

}

Question 8

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*Salar program\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

#include<stdio.h>

#include<conio.h>

int main()

{

int hours=0,ex\_time;

float gp,hourly\_rate,overpay;

while(hours>=0)

{

printf("Enter # hours worked (-1 to end):");

scanf("%d",&hours);

if(hours==-1)

{

printf("Program ends");

break;

}

printf("Enter hourly rate of the worker ($00.00):");

scanf("%f",&hourly\_rate);

if(hours<=40)

{

gp=hours\*hourly\_rate;

printf("Salary is %f\n",gp);

}

else

{

ex\_time=hours-40;

overpay=ex\_time\*(hourly\_rate/2);

gp=hours\*hourly\_rate+overpay;

printf("Salary is %f\n",gp);

}

}

getch();

return 0;

}

Question 9

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*PRE DECREMENT\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

#include<stdio.h>

#include<conio.h>

int main()

{

int a;

//for predecrementing

a=40;

printf("Predecrementing 40:\n");

printf("%d\n",--a);

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

//for postdecrementing

a=40;

printf("Postdecrementing 40:\n");

printf("%d\n",a--);

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

getch();

return 0;

}

Question 10

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*PRE DECREMENT\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

#include<stdio.h>

#include<conio.h>

int main()

{

int a;

//for predecrementing

a=11;

printf("Preincrementing 40:\n");

printf("%d\n",a++);

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

//for postdecrementing

a=11;

printf("Postincrementing 40:\n");

printf("%d\n",a++);

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

getch();

return 0;

}

Question 11

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*DISPLAY THE REQUIRED ARRANGEMENTS\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

#include<stdio.h>

#include<conio.h>

int main()

{

int i;

printf("A\tA+2\tA+4\tA+6\n");

for(i=3;i<=15;i=i+3)

{

printf("%d\t%d\t%d\t%d\n",i,i+2,i+4,i+6);

}

getch();

return 0;

}

Question 12

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*NUMBERS RANKING\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

#include<stdio.h>

#include<conio.h>

int main()

{

int i,num,largest=0,secondlargest=0;

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

{

printf("Enter a number:\n");

scanf("%d",&num);

if(num>largest)

{

secondlargest=largest;

largest=num;

}

else if(num<largest)

{

if(num>secondlargest)

{

secondlargest=num;

}

}

}

printf("The largest number is %d\n",largest);

printf("The second largest number is %d\n",secondlargest);

getch();

return 0;

}

Question 13

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*PAINDROME NUMBER\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

#include<stdio.h>

#include<conio.h>

int main()

{

int num,digit1,digit2,digit3,digit4,digit5;

printf("Enter the five digit number:");

scanf("%d",&num);

digit1=(num/10000);

digit2=(num%10000)/1000;

digit3=(num%1000)/100;

digit4=(num%100)/10;

digit5=num%10;

if(digit1==digit5&&digit2==digit4)

printf("The given number is a palindrome\n");

else

printf("The given number is not a palindrome\n");

getch();

return 0;

}

Question 14

#include<stdio.h>

#include<conio.h>

int main()

{

int num,binary,decimalvalue=0;

int last\_digit,base=1;

printf("Enter the integer containing only 0s and 1s:");

scanf("%d",&num);

binary=num;

while(num)

{

last\_digit=num%10;

decimalvalue=decimalvalue+last\_digit\*base;

num=num/10;

base=base\*2;

}

printf("The decimal equivalent of %d is %d",binary,decimalvalue);

getch();

return 0;

}

Question 15(a)

#include<stdio.h>

#include<conio.h>

int main()

{

int n,fact=1;

printf("Enter a nonnegative integer to find its factorial:");

scanf("%d",&n);

if(n>=0)

{

for(int i=n;i>=1;i--)

{

fact=fact\*i;

}

printf("The factorial of %d is %d",n,fact);

}

getch();

return 0;

}

Question 15(b)

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*FACTORIAL\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*?

#include<stdio.h>

#include<conio.h>

int main()

{

float fact=1,e=1;

for(int i=1;i<=15;i++)

{

fact=fact\*i;

e=e+1/fact; // e=1+1/1!+1/2!+1/3!+...

}

printf("The estimated value of e is %f",e);

getch();

return 0;

}

Question 16

Functions

Definition

A **function** is a block of organized, reusable code that is used to perform a single, related action. **Functions** provide better modularity for your application and a high degree of code reusing. ... Different **programming** languages name them differently, for example, **functions**, methods, sub-routines, procedures

Function are used for **divide a large code into module**, due to this we can easily debug and maintain the code. For example if we write a calculator programs at that time we can write every logic in a separate function (For addition sum(), for subtraction sub()). Any function can be called many times.

**Advantage of Function**

* Code Re-usability
* Develop an application in module format.
* Easily to debug the program.
* **Code optimization:**No need to write lot of code.

**Defining a function.**

Defining of function is nothing but give body of function that means write logic inside function body.

**Syntax**

1. return\_type function\_name(parameter)
2. {
3. function body;
4. }

* **Return type:**A function may return a value. The return\_type is the data type of the value the function returns.Return type parameters and returns statement are optional.
* **Function name:**Function name is the name of function it is decided by programmer or you.
* **Parameters:**This is a value which is pass in function at the time of calling of function A parameter is like a placeholder. It is optional.
* **Function body:**Function body is the collection of statements.

**Function Declarations**

A function declaration is the process of tells the compiler about a function name. The actual body of the function can be defined separately.

**Syntax**

1. return\_type function\_name(parameter);

At the time of function declaration function must be terminated with **;**.

**calling a function.**

When we call any function control goes to function body and execute entire code. For call any function just write name of function and if any parameter is required then pass parameter.

**Syntax**

1. function\_name();
2. or
3. variable=function\_name(argument);

**Example of Function**

#include<stdio.h>

1. #include<conio.h>
3. void sum(); // declaring a function
4. clrsct();
5. int a=10,b=20, c;
7. void sum() // defining function
8. {
9. c=a+b;
10. printf("Sum: %d", c);
11. }
12. void main()
13. {
14. sum(); // calling function
15. }
16. **Output**
17. Sum: 30

**Important points related to Function**

* The basic purpose of the function is code reuse.
* From any function we can invoke (call) any another functions.
* Always compilation will be take place from top to bottom.
* Always execution process will starts from main() and ends with main() only.
* In implementation when we are calling a function which is define later for avoiding the compilation error we need to for forward declaration that is prototype is required.
* In function definition first line is called function declaration or function header.
* Always function declaration should be match with function declaratory.
* In implementation whenever a function does not returns any values back to the calling place then specify the return type.
* Void means nothing that is no return value.
* In implementation whenever a function returns other than void then specify the return type as return value type that is on e type of return value it is returning same type of return statement should be mentioned.
* Default return type of any function is an **int**.
* Default parameter type of any function is **void**.

Reccursion

The process in which a function calls itself directly or indirectly is called recursion and the corresponding function is called as recursive function. Using recursive algorithm, certain problems can be solved quite easily. Examples of such problems are [Towers of Hanoi (TOH)](http://quiz.geeksforgeeks.org/c-program-for-tower-of-hanoi/), [Inorder/Preorder/Postorder Tree Traversals](https://www.geeksforgeeks.org/tree-traversals-inorder-preorder-and-postorder/), [DFS of Graph](https://www.geeksforgeeks.org/depth-first-traversal-for-a-graph/), etc.

* **A Mathematical Interpretation**
* Let us consider a problem that a programmer have to determine the sum of first n natural numbers, there are several ways of doing that but the simplest approach is simply add the numbers starting from 1 to n. So the function simply looks like,
* int fact(int n)
* {
* if (n < = 1) // base case
* return 1;
* else
* return n\*fact(n-1);
* }

Question 17

A storage class represents the visibility and a location of a variable. It tells from what part of code we can access a variable. A storage class in C is used to describe the following things:

* The variable scope.
* The location where the variable will be stored.
* The initialized value of a variable.
* A lifetime of a variable.
* Who can access a variable?

Thus a storage class is used to represent the information about a variable.

NOTE: A variable is not only associated with a data type, its value but also a storage class.

There are total four types of standard storage classes. The table below represents the storage classes in C.

|  |  |
| --- | --- |
| **Storage class** | **Purpose** |
| **Auto** | It is a default storage class. |
| **Extern** | It is a global variable. |
| **Static** | It is a local variable which is capable of returning a value even when control is transferred to the function call. |
| **register** | It is a variable which is stored |

## Auto Storage Class in C

The variables defined using auto storage class are called as local variables. Auto stands for automatic storage class. A variable is in auto storage class by default if it is not explicitly specified.

The scope of an auto variable is limited with the particular block only. Once the control goes out of the block, the access is destroyed. This means only the block in which the auto variable is declared can access it.

A keyword auto is used to define an auto storage class. By default, an auto variable contains a garbage value.

Example, auto int age;

The program below defines a function with has two local variables

int add(void) {

int a=13;

auto int b=48;

return a+b;}

## Extern Storage Class in C

Extern stands for external storage class. Extern storage class is used when we have global functions or variables which are shared between two or more files.

Keyword**extern** is used to declaring a global variable or function in another file to provide the reference of variable or function which have been already defined in the original file.

The variables defined using an extern keyword are called as global variables. These variables are accessible throughout the program. Notice that the extern variable cannot be initialized it has already been defined in the original file.

Example, extern void display();

### First File: main.c

#include <stdio.h>

extern i;

main() {

printf("value of the external integer is = %d\n", i);

return 0;}

### Second File: original.c

#include <stdio.h>

i=48;

Result:

value of the external integer is = 48

## Static Storage Class in C

The static variables are used within function/ file as local static variables. They can also be used as a [global variable](https://www.guru99.com/local-vs-global-variable.html)

* Static local variable is a local variable that retains and stores its value between function calls or block and remains visible only to the function or block in which it is defined.
* Static global variables are global variables visible **only to the file in which it is declared.**

Example: static int count = 10;

Keep in mind that static variable has a default initial value zero and is initialized only once in its lifetime.

#include <stdio.h> /\* function declaration \*/

void next(void);

static int counter = 7; /\* global variable \*/

main() {

while(counter<10) {

next();

counter++; }

return 0;}

void next( void ) { /\* function definition \*/

static int iteration = 13; /\* local static variable \*/

iteration ++;

printf("iteration=%d and counter= %d\n", iteration, counter);}

Result:

iteration=14 and counter= 7

iteration=15 and counter= 8

iteration=16 and counter= 9

Global variables are accessible throughout the file whereas static variables are accessible only to the particular part of a code.

The lifespan of a static variable is in the entire program code. A variable which is declared or initialized using static keyword always contains zero as a default value.

## Register Storage Class in C

You can use the register storage class when you want to store local variables within functions or blocks in CPU registers instead of RAM to have quick access to these variables. For example, "counters" are a good candidate to be stored in the register.

Example: register int age;

The keyword **register** is used to declare a register storage class. The variables declared using register storage class has lifespan throughout the program.

It is similar to the auto storage class. The variable is limited to the particular block. The only difference is that the variables declared using register storage class are stored inside CPU registers instead of a memory. Register has faster access than that of the main memory.

The variables declared using register storage class has no default value. These variables are often declared at the beginning of a program.

#include <stdio.h> /\* function declaration \*/

main() {

{register int weight;

int \*ptr=&weight ;/\*it produces an error when the compilation occurs ,we cannot get a memory location when dealing with CPU register\*/}

}

OUTPUT:

error: address of register variable 'weight' requested

The next table summarizes the principal features of each storage class which are commonly used in C programming

Question 18

Built in and user defined functions

The C language doesn’t define any built-in functions.

It does provide a number of standard library functions, such as printf. These functions are typically implemented the same way as ordinary C functions, but they’re provided as part of the C implementation.

A built-in function would be one that’s implemented directly by the compiler. For example, if you look at the generated code for a call to a user-defined function, you’ll see a CALL instruction, or whatever instruction is used by the target CPU. For a built-in function, you might see some other code sequence; for example something that looks like a call to sqrt might result in a SQRT instruction (if the CPU supports it).

But there’s really no fundamental difference as far as the language is concerned. A compiler can implement any function any way it likes, as long as the resulting behavior is *as if* it were implemented as a user-defined function.

Some compilers provide *intrinsic* functions as a compiler extension.

Question 19

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*Base Power by use of fuctions\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

#include<stdio.h>

#include<conio.h>

int integerPower(int base,int exponent);

int main()

{

int base,exponent;

printf("Enter the Base:");

scanf("%d",&base);

printf("Enter the exponent:");

scanf("%d",&exponent);

printf("%d power %d = %d",base,exponent,integerPower(base,exponent));

getch();

return 0;

}

int integerPower(int base,int exponent)

{

int p=1;

for(int i=1;i<=exponent;i++)

{

p=p\*base;

}

return p;

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* base power\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

//Two numbers are entered through keyboard .Write a program to find value of number raised to power of another

#include<conio.h>

#include<stdio.h>

#include<math.h>

int main()

{

int base,power;

printf("\tEnter Base:");

scanf("%d",&base);

printf("\tEnter Power:");

scanf("%d",&power);

int num;

num=pow(base,power);

printf("%d power %d =",base,power);

printf("%d",num);

printf("\n\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_");

printf("\n\n \t\t%d ^ %d = %d",base,power,num);

getch();

return 0;

}//main closing

Question 20

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

A PROGRAM THAT INPUT A SERIES OF INTEGERS AND PASSES THEM ONE AT A

TIME TO FUNCTION EVEN, WHICH USES THE REINDER OPERATOR TO DETERINE

IF AN INTEGER IS EVEN THE FUNCTION TAKE INTEGER ARGUENTS AND RETURN 1 IF INTEGER IS EVEN, OR RETURN O IF INTEGER IS ODD

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

#include<stdio.h>

#include<conio.h>

int num;

int even(int a)

{

if(num%2==0)

return 1;

if(num%2!=0)

return 0;

}

int main()

{

char choice;

do

{

printf("\tEnter integers to check even or odd\n\t");

scanf("%d",&num);

printf("%d",even(num));

printf("\nYou want to enter more integers\t(Y/N)\n\t");

fflush(stdin);

scanf("%c",&choice);

}while(choice=='Y');

return 0;

getch();

}

Question 21

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

A PROGRAM THAT RETURNS THE SALLEST OF THREE FLOATING POINT NUMBERS

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

#include<stdio.h>

#include<conio.h>

int num;

int smallest(int small)

{

int a ,b,c; // variable declaration

printf("Enter 3 floating point values:\n");

scanf("%f %f %f",&a,&b,&c); // taking user input for floatng point values

int sm= (

(((a<b)&&(a<c))

? (printf("a (%f) is smallest number",a))

:((b<c)&&(b<a))?(printf("b ( %f ) is smallest number",b)) : printf("c (%f) is sallest number",c))

) ;

return sm;

}

int main()

{

int val;

printf("%c",smallest(val));

return 0;

getch();

}

Question 22

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*A program that display perfect numbers between 1 and 1000\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

#include <stdio.h>

/\* Function declarations \*/

int checkPerfect(int n1);

void PerfectNumbers(int startLimit, int endLimit);

int main()

{

int startLimit=1, endLimit=1000;

printf("\n\n Function : perfect numbers in a given range :\n");

printf("--------------------------------------------------\n");

printf("\nlowest search limit of perfect numbers is %d : ",startLimit);

printf("\nHighest search limit of perfect numbers is %d : ",endLimit);

printf("\n The perfect numbers between %d to %d are : \n", startLimit, endLimit);

PerfectNumbers(startLimit, endLimit);

printf("\n\n");

return 0;

}

// Checks whether the given number is perfect or not.

int checkPerfect(int n1)

{

int i, sum;

sum = 0;

for(i=1; i<n1; i++)

{

if(n1 % i == 0)

{

sum += i;

}

}

// If sum of proper positive divisors equals to given number

// then the number is perfect number

if(sum == n1)

return 1;

else

return 0;

}

void PerfectNumbers(int startLimit, int endLimit)

{

/\* print perfect numbers from start to end \*/

while(stLimit <= endLimit)

{

if(checkPerfect(startLimit))

{

printf(" %d ", startLimit);

}

startLimit++;

}

}

Question 22(b)

/

/\* \*\*\*\*\*\*\*\*\*\*\*\*\*\*write a function which determines if a number is prime or not\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

#include<stdio.h>

int check\_prime(int);

main()

{

int n, result;

printf("Enter an integer to check whether it is prime or not.\n");

scanf("%d",&n);

result = check\_prime(n);

if ( result == 1 )

printf("%d is prime.\n", n);

else

printf("%d is not prime.\n", n);

return 0;

}

int check\_prime(int a)

{

int c;

for ( c=2;c<=a-1;c++)

{

if ( a%c == 0 )

return 0;

}

return 1;

}

Question 23

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* C program to list all prime number between an interval using function.

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

#include <stdio.h>

/\* Function declarations \*/

int isPrime(int num);

void printPrimes(int lowerLimit, int upperLimit);

int main()

{

int lowerLimit=1, upperLimit=50;

// Call function to print all primes between the given range.

printPrimes(lowerLimit, upperLimit);

return 0;

}

void printPrimes(int lowerLimit, int upperLimit)

{

printf("All prime number between %d to %d are: ", lowerLimit, upperLimit);

while(lowerLimit <= upperLimit)

{

// Print if current number is prime.

if(isPrime(lowerLimit))

{

printf("\n");

printf("\t %d ", lowerLimit);

}

lowerLimit++;

}

}

/\*

\* Check whether a number is prime or not.

\* Returns 1 if the number is prime otherwise 0.

\*/

int isPrime(int num)

{

int i;

for(i=2; i<=num/2; i++)

{

/\*

\* If the number is divisible by any number

\* other than 1 and self then it is not prime

\*/

if(num % i == 0)

{

return 0;

}

}

return 1;

}

Question 24

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Write a program that takes an integer value and returns the number with its digits reversed

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

#include<conio.h>

#include<stdio.h>

int main()

{

int num;

printf("\tEnter 5 digit number:\n\t");

scanf("%d", &num);

printf("\t---------------------\n");

printf("\tReversed 5 digit number:\n\t");

int num1=num%10;

printf("\t%d",num1);

int num2 =(num/10)%10;

printf("%d",num2);

int num3 =((num/10)/10)%10;

printf("%d",num3);

int num4 =(((num/10)/10)/10)%10;

printf("%d",num4);

int num5 =((((num/10)/10)/10)/10)%10;

printf("%d",num5);

getch();

return 0;

}

Question 25

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*Guess Game\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

#include <stdlib.h>

#include <conio.h>

#include <stdio.h>

int high=1000,low=1;

int n,secnum;

int guess(int k)

{

secnum = rand()%(high-low)+low;

if(n>secnum)

{

printf("\nOops! Guess was too high ");

printf("\n\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\n\t");

}

else if(n<secnum)

{

printf("\nOops! Guess was too low");

printf("\n\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\n\t");

}

else if(n == secnum)

{

printf("\n\t\tHurrah! You win");

printf("\n\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\n\t");

}

}

int main()

{

char ch;

printf("WELLCOME TO THE GUESS GAME %c",1);

printf("\n \tI have a number between 1 to 1000\n \t can you guess my number\n");

printf("\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\n\t");

printf("\nPlease type your first guess\n \t===");

scanf("%d",&n);

printf("\nExcellent you make a guess");

printf("%d",guess (n));

printf("\nWould you like to try again?\n\t(y/n)\_\_\_\_\_\_\_\_");

fflush(stdin);

scanf("%c",&ch);

while(ch=='y')

{

do{

printf("\n\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\n\t");

printf("\n Make a guess===");

scanf("%d",&n);

printf("%d",guess (n));

printf("\nWould you like to try again?\n\t(y/n)\_\_\_\_\_\_\_\_\_\n");

fflush(stdin);

scanf("%c",&ch);

} while(ch=='y');

}

printf("\n\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\n\t");

printf("\nSecret Number was %d",secnum);

getch();

return 0;

}

Question 26(a)

//Fibonacci series program in C

#include <stdio.h>

int n=50, first = 0, second = 1, next, c;

int fab()

{

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

{

if (c <= 1)

next = c;

else

{ next = first + second;

first = second;

second = next;

}

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

}

}

int main()

{

printf("First %d terms of Fibonacci series are:\n", n);

printf("%d",fab());

return 0;

}

\* Enter terms: 10

\* 0 1 1 2 3 5 8 13 21 34

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

Question 26(b)

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Program to print Fibonacci Sequence using recursion

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

#include<stdio.h>

int num=0 , terms=50;

int fibonacci();

int series()

{

if( num < terms)

printf("%d ", fibonacci());

}

int main(void)

{

printf(" 50 terms of Fabonaccii series: ");

printf("%d",series);

return 0; // return 0 to operating system

}

int fibonacci(num)

{

if(num == 0 || num == 1)

{

return num;

}

else

{

// recursive call

return fibonacci(num-1) + fibonacci(num-2);

}

}

Question 28

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* TEST PALINDROME \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

#include <string.h>

#include <conio.h>

#include<stdio.h>

int checkpalindrome(char \*s)

{

static int i,c=0,n=strlen(s);

if(i<n/2)

{

if(s[i]==s[n-i-1])

c++;

i++;

checkpalindrome(s);

}

else

{

if(c==i)

return 1;

else

return 0;

}

}

int main()

{

char s[1000];

printf("Enter the string: \n\t");

gets(s);

if(checkpalindrome(s)==1)

printf("string is palindrome");

else

printf("string is not palindrome");

}

Question 29

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* print array\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

#include <stdio.h>

#include<stdlib.h>

#define MAX\_SIZE 100

/\* Function declaration \*/

int start, len;

int printArray(int arr[], int, int);

int main()

{

int arr[MAX\_SIZE];

int N, i;

/\* Input size and elements in array \*/

printf("Enter size of the array: ");

scanf("%d", &N);

/\* Prints array recursively \*/

if(arr[i]==0)

{

exit(0);

}

if(arr[i]!=0)

{

printf("Enter elements in the array: ");

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

{

scanf("%d", &arr[i]);

}

printf("Elements in the array: ");

printArray(arr, 0, N);

/\* Prints the current array element \*/

printf("%d, ", arr[start]);

/\* Recursively call printArray to print next element in the array \*/

printArray(arr, start + 1, len);

}

return 0;

}

/\*\*

\* Prints an array recursively within a given range.

\*/

int printArray(int arr[], int start,int len)

{

if(start==0)

return 0;

/\* Recursion base condition \*/

else if(start >= len)

return 1;

}

Question 30

Pointers

The **Pointer** in C, is a variable that stores address of another variable. A pointer can also be used to refer to another pointer function. A pointer can be incremented/decremented, i.e., to point to the next/ previous memory location. The purpose of pointer is to save memory space and achieve faster execution time.

Features and Use of Pointers in C/C++

[Pointers](https://www.geeksforgeeks.org/pointers-in-c-and-c-set-1-introduction-arithmetic-and-array/) store address of variables or a memory location.

**Syntax:**

datatype \*var\_name;

**Example:** pointer “ptr” holds address of an integer variable or holds address of a memory whose value(s) can be accessed as integer values through “ptr”

int \*ptr;

**Features of Pointers:**

1. Pointers save memory space.
2. Execution time with pointers is faster because data are manipulated with the address, that is, direct access to  
   memory location.
3. Memory is accessed efficiently with the pointers. The pointer assigns and releases the memory as well. Hence it can be said the Memory of pointers is dynamically allocated.
4. Pointers are used with data structures. They are useful for representing two-dimensional and multi-dimensional  
   arrays.
5. An array, of any type can be accessed with the help of pointers, without considering its subscript range.
6. Pointers are used for file handling.
7. Pointers are used to allocate memory dynamically.
8. In C++, a pointer declared to a base class could access the object of a derived class. However, a pointer to a derived class cannot access the object of a base class.

**Uses of pointers:**

1. To [pass arguments by reference](https://www.geeksforgeeks.org/passing-by-pointer-vs-passing-by-reference-in-c/)
2. For [accessing array elements](https://www.geeksforgeeks.org/arrays-in-c-cpp/)
3. To [return multiple values](https://www.geeksforgeeks.org/how-to-return-multiple-values-from-a-function-in-c-or-cpp/)
4. [Dynamic memory allocation](https://www.geeksforgeeks.org/dynamic-memory-allocation-in-c-using-malloc-calloc-free-and-realloc/)
5. To [implement data structures](https://www.geeksforgeeks.org/data-structures/)
6. To do [system level programming](https://www.geeksforgeeks.org/tag/system-programming/) where memory addresses are usef

## Types of Pointers in C

Following are the different **Types of Pointers in C**:

### Null Pointer

We can create a null pointer by assigning null value during the pointer declaration. This method is useful when you do not have any address assigned to the pointer. A null pointer always contains value 0.

Following program illustrates the use of a null pointer:

#include <stdio.h>

int main()

{

int \*p = NULL; //null pointer

printf(“The value inside variable p is:\n%x”,p);

return 0;

}

Output:

The value inside variable p is:

0

### Void Pointer

In [C programming](https://www.guru99.com/c-programming-tutorial.html), a void pointer is also called as a generic pointer. It does not have any standard data type. A void pointer is created by using the keyword void. It can be used to store an address of any variable.

Following program illustrates the use of a void pointer:

#include <stdio.h>

int main()

{

void \*p = NULL; //void pointer

printf("The size of pointer is:%d\n",sizeof(p));

return 0;

}

Output:

The size of pointer is:4

### Wild pointer

A pointer is said to be a wild pointer if it is not being initialized to anything. These types of C pointers are not efficient because they may point to some unknown memory location which may cause problems in our program and it may lead to crashing of the program. One should always be careful while working with wild pointers.

Following program illustrates the use of wild pointer:

#include <stdio.h>

int main()

{

int \*p; //wild pointer

printf("\n%d",\*p);

return 0;

}

Output

timeout: the monitored command dumped core

sh: line 1: 95298 Segmentation fault timeout 10s main

Other types of pointers in 'c' are as follows:

* Dangling pointer
* Complex pointer
* Near pointer
* Far pointer
* Huge pointer

## Direct and Indirect Access Pointers

In C, there are two equivalent ways to access and manipulate a variable content

* Direct access: we use directly the variable name
* Indirect access: we use a pointer to the variable

Let's understand this with the help of program below

#include <stdio.h>

/\* Declare and initialize an int variable \*/

int var = 1;

/\* Declare a pointer to int \*/

int \*ptr;

int main( void )

{

/\* Initialize ptr to point to var \*/

ptr = &var;

/\* Access var directly and indirectly \*/

printf("\nDirect access, var = %d", var);

printf("\nIndirect access, var = %d", \*ptr);

/\* Display the address of var two ways \*/

printf("\n\nThe address of var = %d", &var);

printf("\nThe address of var = %d\n", ptr);

/\*change the content of var through the pointer\*/

\*ptr=48;

printf("\nIndirect access, var = %d", \*ptr);

return 0;}

After compiling the program without any errors, the result is:

Direct access, var = 1

Indirect access, var = 1

The address of var = 4202496

The address of var = 4202496

Indirect access, var = 48

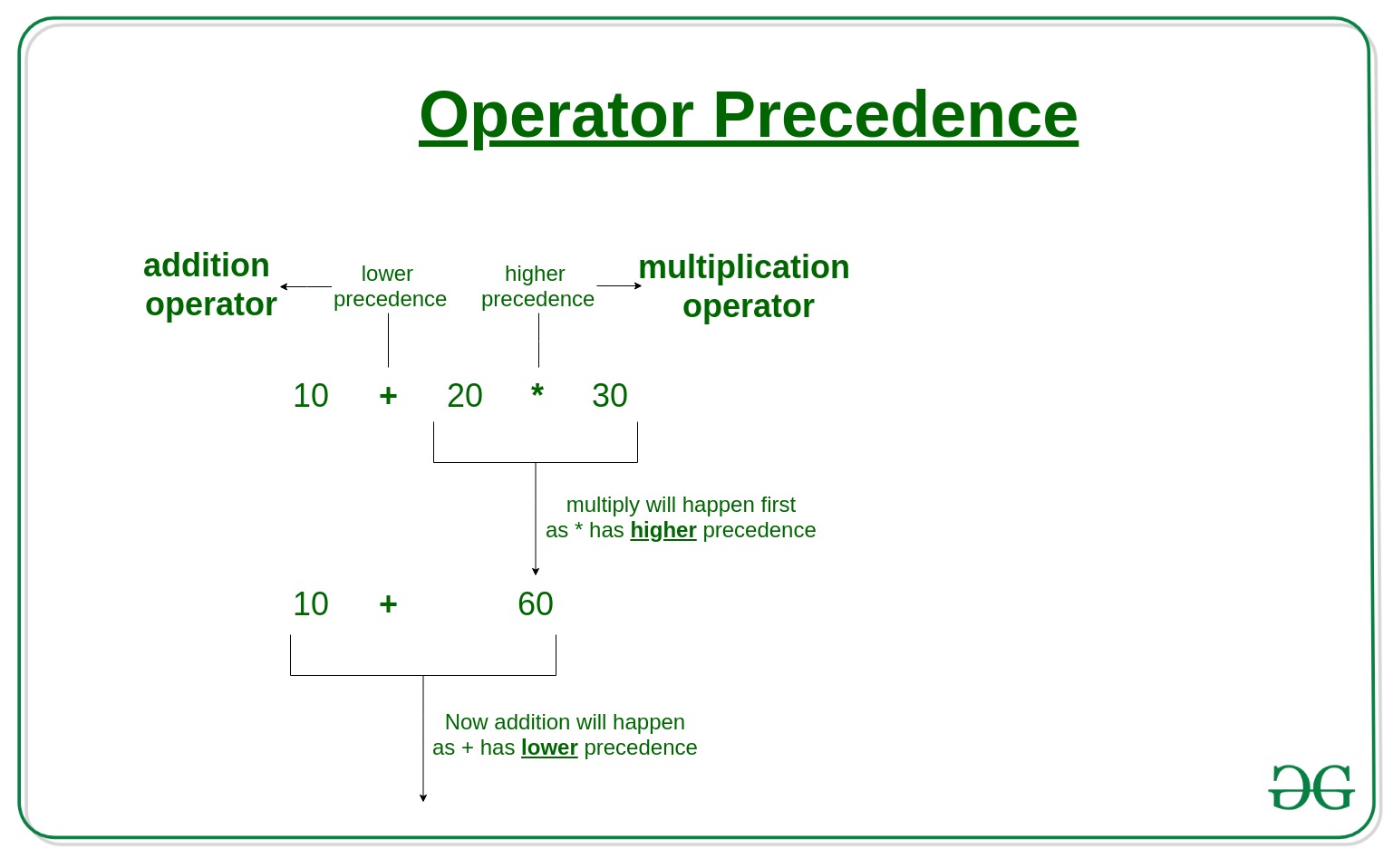
Question 31

**Operator precedence**

 Determines which operator is performed first in an expression with more than one operators with different precedence.

**For example:** Solve

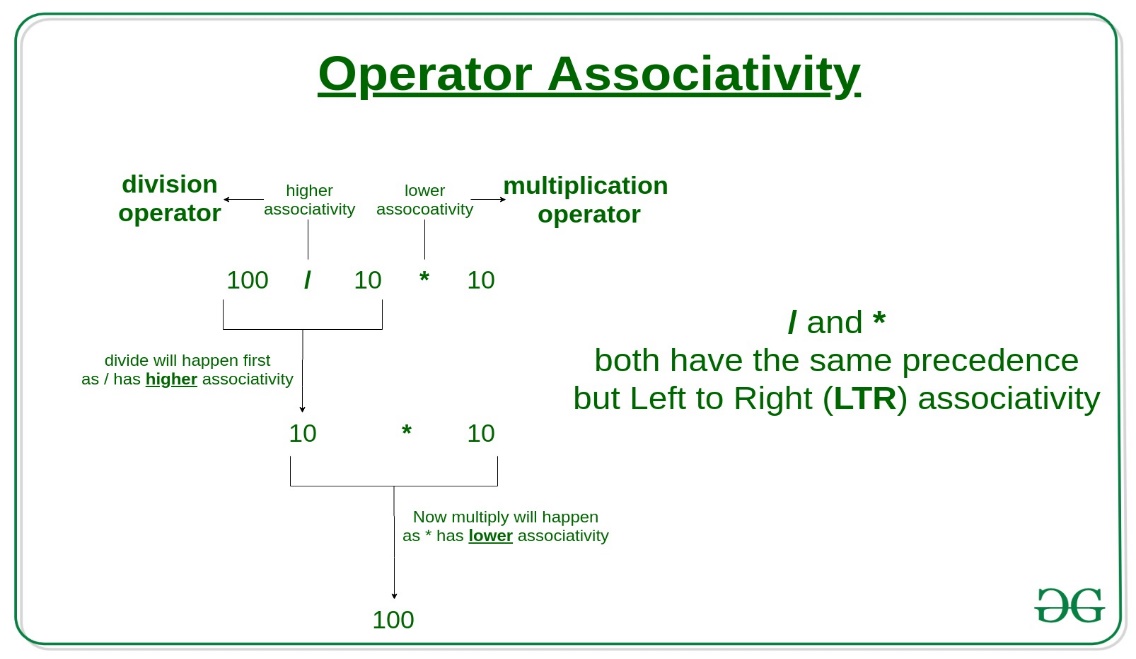
10 + 20 \* 30



**Operators Associativity**

 is used when two operators of same precedence appear in an expression. Associativity can be either **L**eft**t**o **R**ight or**R**ight**t**o **L**eft.

**For example:** ‘\*’ and ‘/’ have same precedence and their associativity is **L**eft**t**o **R**ight, so the expression “100 / 10 \* 10” is treated as “(100 / 10) \* 10”.



Features

**1) Associativity is only used when there are two or more operators of same precedence.**  
The point to note is associativity doesn’t define the order in which operands of a single operator are evaluated. For example, consider the following program, associativity of the + operator is left to right, but it doesn’t mean f1() is always called before f2(). The output of the following program is in-fact compiler dependent. See [this](https://www.geeksforgeeks.org/evaluation-order-of-operands/) for details.

filter\_none

edit

play\_arrow

brightness\_4

|  |
| --- |
| // Associativity is not used in the below program.  // Output is compiler dependent.    #include <stdio.h>    int x = 0;  int f1()  {      x = 5;      return x;  }  int f2()  {      x = 10;      return x;  }  int main()  {      int p = f1() + f2();      printf("%d ", x);      return 0;  } |

**2) All operators with the same precedence have same associativity**  
This is necessary, otherwise, there won’t be any way for the compiler to decide evaluation order of expressions which have two operators of same precedence and different associativity. For example + and – have the same associativity.

**3) Precedence and associativity of postfix ++ and prefix ++ are different**  
Precedence of postfix ++ is more than prefix ++, their associativity is also different. Associativity of postfix ++ is left to right and associativity of prefix ++ is right to left. See [this](https://www.geeksforgeeks.org/difference-between-p-p-and-p/) for examples.

**4) Comma has the least precedence among all operators and should be used carefully**

 For example consider the following program, the output is 1. See [this](https://www.geeksforgeeks.org/a-comma-operator-question/) and [this](https://www.geeksforgeeks.org/comna-in-c-and-c/) for more details.

|  |  |  |
| --- | --- | --- |
| **Category** | **Operator** | **Associativity** |
| Postfix | () [] -> . ++ - - | Left to right |
| Unary | + - ! ~ ++ - - (type)\* & sizeof | Right to left |
| Multiplicative | \* / % | Left to right |
| Additive | + - | Left to right |
| Shift | << >> | Left to right |
| Relational | < <= > >= | Left to right |
| Equality | == != | Left to right |
| Bitwise AND | & | Left to right |
| Bitwise XOR | ^ | Left to right |
| Bitwise OR | | | Left to right |
| Logical AND | && | Left to right |
| Logical OR | || | Left to right |
| Conditional | ?: | Right to left |
| Assignment | = += -= \*= /= %=>>= <<= &= ^= |= | Right to left |
| Comma | , | Left to right |

Question 32

## #include preprocessor directive (File inclusion directive)

We use #include directive to include contents of another file (especially header file's) to a program. We can include a file in two different ways.

1. **#include <file.h>**  
   This variant of including header file, searches header file in the "standard header file directory". Standard header file directory is the path where all header files are stored.

**Syntax to use**

#include <header.h>

Where header.h is a standard C header file.

1. **#include "file"**  
   We use this variant of including file when we want to include our own/custom header file. It searches the file in the current directory and then in the standard header file’s directory.

**Syntax to use**

#include "path\_to\_header\_file";

Where path\_to\_header\_file is relative path to the header file.

**Example:**

#include <stdio.h>

#include "main.h"

#include "win/display.c"

Here, file "stdio.h" is a standard header file, "main.h" and "win/display.c" is custom

## #define preprocessor directive

#define preprocessor directive is the most useful preprocessor directive in C language. We use it to define a name for particular value/constant/expression.

C preprocessor processes the defined name and replace each occurrence of a particular string/defined name (macro name) with a given value (micro body).

**Syntax to define a MACRO using #define**

#define MACRO\_NAME macro\_body

Here, MACRO\_NAME is

a [C identifier](https://codeforwin.org/2017/08/keywords-identifiers-c.html) and macro\_body is string/value/expression.

#include <stdio.h>

// Macro definition

#define COUNTRY "INDIA" // String constant

#define TRUE 1 // Integer constant

#define FALSE 0 // Integer constant

#define SUM (10 + 20) // Macro definition

int main()

{

printf("COUNTRY: %s\n", COUNTRY);

printf("TRUE: %d\n", TRUE);

printf("FALSE: %d\n", FALSE);

printf("SUM(10 + 20): %d\n", SUM);

return 0;

}

**Output:**

COUNTRY: INDIA

TRUE: 1

FALSE: 0

SUM(10 + 20): 30

Here, COUNTRY, TRUE and FALSE are compile time constants and SUM is a macro. The C pre-processor replaces all occurrences of COUNTRY with "India" and all other constants with their respective value before compilation

Question 33

Ctype.h

The **ctype.h** header file of the C Standard Library declares several functions that are useful for testing and mapping characters.

All the functions accepts **int** as a parameter, whose value must be EOF or representable as an unsigned char.

All the functions return non-zero (true) if the argument c satisfies the condition described, and zero(false) if not.

Following are the functions defined in the header ctype.h −

|  |  |
| --- | --- |
| **Sr.No.** | **Function & Description** |
| 1 | [int isalnum(int c)](https://www.tutorialspoint.com/c_standard_library/c_function_isalnum.htm)  This function checks whether the passed character is alphanumeric. |
| 2 | [int isalpha(int c)](https://www.tutorialspoint.com/c_standard_library/c_function_isalpha.htm)  This function checks whether the passed character is alphabetic. |
| 3 | [int iscntrl(int c)](https://www.tutorialspoint.com/c_standard_library/c_function_iscntrl.htm)  This function checks whether the passed character is control character. |
| 4 | [int isdigit(int c)](https://www.tutorialspoint.com/c_standard_library/c_function_isdigit.htm)  This function checks whether the passed character is decimal digit. |
| 5 | [int isgraph(int c)](https://www.tutorialspoint.com/c_standard_library/c_function_isgraph.htm)  This function checks whether the passed character has graphical representation using locale. |
| 6 | [int islower(int c)](https://www.tutorialspoint.com/c_standard_library/c_function_islower.htm)  This function checks whether the passed character is lowercase letter. |
| 7 | [int isprint(int c)](https://www.tutorialspoint.com/c_standard_library/c_function_isprint.htm)  This function checks whether the passed character is printable. |
| 8 | [int ispunct(int c)](https://www.tutorialspoint.com/c_standard_library/c_function_ispunct.htm)  This function checks whether the passed character is a punctuation character. |
| 9 | [int isspace(int c)](https://www.tutorialspoint.com/c_standard_library/c_function_isspace.htm)  This function checks whether the passed character is white-space. |
| 10 | [int isupper(int c)](https://www.tutorialspoint.com/c_standard_library/c_function_isupper.htm)  This function checks whether the passed character is an uppercase letter. |
| 11 | [int isxdigit(int c)](https://www.tutorialspoint.com/c_standard_library/c_function_isxdigit.htm)  This function checks whether the passed character is a hexadecimal digit |

**C – stdio.h**

All C inbuilt functions which are declared in stdio.h header file are given below.

**List of inbuilt C functions in stdio.h file:**

1. **printf()** This function is used to print the character, string, float, integer, octal and hexadecimal values onto the output screen
2. **scanf()** This function is used to read a character, string, numeric data from keyboard.
3. **getc()** It reads character from file
4. **gets()** It reads line from keyboard
5. **getchar()** It reads character from keyboard
6. **puts()** It writes line to o/p screen
7. **putchar()** It writes a character to screen
8. **clearerr()** This function clears the error indicators
9. **f open()** All file handling functions are defined in stdio.h header file
10. **f close()** closes an opened file
11. **getw()** reads an integer from file
12. **putw()** writes an integer to file
13. **f getc()** reads a character from file
14. **putc()** writes a character to file
15. **f putc()** writes a character to file
16. **f gets()** reads string from a file, one line at a time
17. **f puts()** writes string to a file
18. **f eof()** finds end of file
19. **f getchar** reads a character from keyboard
20. **f getc()** reads a character from file
21. **f printf()** writes formatted data to a file
22. **f scanf()** reads formatted data from a file
23. **f getchar** reads a character from keyboard
24. **f putchar** writes a character from keyboard
25. **f seek()** moves file pointer position to given location
26. **SEEK\_SET** moves file pointer position to the beginning of the file
27. **SEEK\_CUR** moves file pointer position to given location
28. **SEEK\_END** moves file pointer position to the end of file.
29. **f tell()** gives current position of file pointer
30. **rewind()** moves file pointer position to the beginning of the file
31. **putc()** writes a character to file
32. **sprint()** writes formatted output to string
33. **sscanf()** Reads formatted input from a string
34. **remove()** deletes a file
35. **fflush()** flushes a file

String.h

C programming language provides a set of pre-defined functions called **string handling functions** to work with string values. The string handling functions are defined in a header file called **string.h**. Whenever we want to use any string handling function we must include the header file called **string.h**.

The following table provides most commonly used string handling function and their use...

| **Function** | **Syntax (or) Example** | **Description** |
| --- | --- | --- |
| **strcpy()** | strcpy(string1, string2) | Copies string2 value into string1 |
| **strncpy()** | strncpy(string1, string2, 5) | Copies first 5 characters string2 into string1 |
| **strlen()** | strlen(string1) | returns total number of characters in string1 |
| **strcat()** | strcat(string1,string2) | Appends string2 to string1 |
| **strncat()** | strncpy(string1, string2, 4) | Appends first 4 characters of string2 to string1 |
| **strcmp()** | strcmp(string1, string2) | Returns 0 if string1 and string2 are the same; less than 0 if string1<string2; greater than 0 if string1>string2 |
| **strncmp()** | strncmp(string1, string2, 4) | Compares first 4 characters of both string1 and string2 |
| **strcmpi()** | strcmpi(string1,string2) | Compares two strings, string1 and string2 by ignoring case (upper or lower) |
| **stricmp()** | stricmp(string1, string2) | Compares two strings, string1 and string2 by ignoring case (similar to strcmpi()) |
| **strlwr()** | strlwr(string1) | Converts all the characters of string1 to lower case. |
| **strupr()** | strupr(string1) | Converts all the characters of string1 to upper case. |
| **strdup()** | string1 = strdup(string2) | Duplicated value of string2 is assigned to string1 |
| **strchr()** | strchr(string1, 'b') | Returns a pointer to the first occurrence of character 'b' in string1 |
| **strrchr()** | 'strrchr(string1, 'b') | Returns a pointer to the last occurrence of character 'b' in string1 |
| **strstr()** | strstr(string1, string2) | Returns a pointer to the first occurrence of string2 in string1 |
| **strset()** | strset(string1, 'B') | Sets all the characters of string1 to given character 'B'. |
| **strnset()** | strnset(string1, 'B', 5) | Sets first 5 characters of string1 to given character 'B'. |
| **strrev()** | strrev(string1) | It reverses the value of string1 |

Question 34

# **Type**

“A type refers to an extensive system used for declaring variables or functions of different types. The type of a variable determines how much space it occupies in storage and how the bit pattern stored is interpreted.”

Let’s say that every donut in the image above is a type. The chocolate one is an **int**, the one with the sprinkles are **unsigned int**, the one with white icing and sprinkles is a **char**, and so on… Every single one is different in some way from another one. The following table provides details of some of the standard types.

Enumeration

Enumeration is a user defined datatype in C language. It is used to assign names to the integral constants which makes a program easy to read and maintain. The keyword “enum” is used to declare an enumeration.

Here is the syntax of enum in C language,

enum enum\_name{const1, const2, ....... };

The enum keyword is also used to define the variables of enum type. There are two ways to define the variables of enum type as follows.

enum week{sunday, monday, tuesday, wednesday, thursday, friday, saturday};

enum week day;

Structure

A data structure is just the organization of data, in a particular way that can be used efficiently. Think of it as the deck of cards. Each card is an element from the structure (deck), in which the elements (card) has a given value. Something like this:

typedef struct s\_deck  
{   
 card2 = 2;  
 card3 = 3;  
 ...   
 cardKing = 13;  
 cardAS = 14;  
}

I know that each card is unique from each other in some way, such like 8th of Spade and 8th of Hearts; they have the same value, but different name, color, etc… For the sake of this tutorial, we just gonna cared about the value of the card (number)

Question 35

## **File Handling In C++**

Files are used to store data in a storage device permanently. File handling provides a mechanism to store the output of a program in a file and to perform various operations on it.

A stream is an abstraction that represents a device on which operations of input and output are performed. A stream can be represented as a source or destination of characters of indefinite length depending on its usage.

In C++ we have a set of file handling methods. These include ifstream, ofstream, and fstream. These classes are derived from fstrembase and from the corresponding iostream class. These classes, designed to manage the disk files, are declared in fstream and therefore we must include fstream and therefore we must include this file in any program that uses files.

In C++, files are mainly dealt by using three classes fstream, ifstream, ofstream.

* ofstream: This Stream class signifies the output file stream and is applied to create files for writing information to files
* ifstream: This Stream class signifies the input file stream and is applied for reading information from files
* fstream: This Stream class can be used for both read and write from/to files.

All the above three classes are derived from fstreambase and from the corresponding iostream class and they are designed specifically to manage disk files.  
C++ provides us with the following operations in File Handling:

* Creating a file: open()
* Reading data: read()
* Writing new data: write()
* Closing a file: close()

Moving on with article on File Handling in C++

## **Opening a File**

Generally, the first operation performed on an object of one of these classes is to associate it to a real file. This procedure is known to open a file.

We can open a file using any one of the following methods:  
1. First is bypassing the file name in constructor at the time of object creation.  
2. Second is using the open() function.

|  |  |
| --- | --- |
| 1 | open() function |

To open a file use

|  |  |
| --- | --- |
| 1 | void open(const char\* file\_name,ios::openmode mode); |

**Syntax**

Here, the first argument of the open function defines the name and format of the file with the address of the file.

The second argument represents the mode in which the file has to be opened. The following modes are used as per the requirements.

|  |  |
| --- | --- |
| ***Modes*** | ***Description*** |
| in | Opens the file to read(default for ifstream) |
| out | Opens the file to write(default for ofstream) |
| binary | Opens the file in binary mode |
| app | Opens the file and appends all the outputs at the end |
| ate | Opens the file and moves the control to the end of the file |
| trunc | Removes the data in the existing file |
| nocreate | Opens the file only if it already exists |
| noreplace | Opens the file only if it does not already exist |

**Example**

|  |  |
| --- | --- |
| 1  2 | fstream new\_file;  new\_file.open(“newfile.txt”, ios::out); |

In the above example, new\_file is an object of type fstream, as we know fstream is a class so we need to create an object of this class to use its member functions. So we create new\_file object and call open() function. Here we use out mode that allows us to open the file to write in it.

Default Open Modes :

* ifstream ios::in
* ofstream ios::out
* fstream ios::in | ios::out

We can combine the different modes using or symbol | .

**Example**

ofstream new\_file;

|  |  |
| --- | --- |
| 1 | new\_file.open(“new\_file.txt”, ios::out | ios::app ); |

Here, input mode and append mode are combined which represents the file is opened for writing and appending the outputs at the end.

As soon as the program terminates, the memory is erased and frees up the memory allocated and closes the files which are opened.  
But it is better to use the close() function to close the opened files after the use of the file.

Using a stream insertion operator << we can write information to a file and using stream extraction operator >> we can easily read information from a file

# File Handling in C – An Easy Concept to Manage your Files in C

Sometimes, you might have felt that managing different types of files in C gets a bit complicated, **whether** it be a text file or a binary file. But, this task proves to be quite simple when we talk about file handling in C.

File handling in C refers to the task of storing data in the form of input or output produced by running C programs in data files, namely, a text file or a binary file for future reference and analysis.

In this File Handling in C tutorial, we will discuss:

Before we begin, let us acknowledge the **significance of file handling**.

Once we compile and run the program, the output is obtained, but this output is not stored in the form of information anywhere in the system.

What if we want to store the output produced for future references? After all, most of the software firms write programs in order to store the output produced as information. This problem can easily be solved by the implementation of file handling in C. Since most of the computer systems work with files as it helps in storing information, C offers this benefit of file handling.

## **1. What is File Handling in C?**

A file is nothing but a source of storing information permanently in the form of a sequence of bytes on a disk. The contents of a file are not volatile like the C compiler memory. The various operations available like creating a file, opening a file, reading a file or manipulating data inside a file is referred to as file handling.

***?***

## **2. Need for File Handling in C**

There is a time when the output generated by compiling and running the program does not serve the purpose. If we want to check the output of the program several times, it becomes a tedious task to compile and run the same program multiple times. This is where file handling comes into play. Here are some of the following reasons behind the popularity of file handling:

* **Reusability:** It helps in preserving the data or information generated after running the program.
* **Large storage capacity:** Using files, you need not worry about the problem of storing data in bulk.
* **Saves time:** There are certain programs that require a lot of input from the user. You can easily access any part of the code with the help of certain commands.
* **Portability:** You can easily transfer the contents of a file from one computer system to another without having to w

Question 36

#include <stdio.h>

int main()

{

char name[50],Father\_Name[20],subj[20];

int Roll\_num,i,n=10,clas;

printf("Record of 10 students: ");

FILE \*fptr;

fptr=(fopen("C:\\student.txt","w"));

for (i=1;i<=n;++i) {

printf("\n For student %d \n\t\tEnter name: ",i+1);

fflush(stdin);

scanf("%s",&name);

printf("\t\tEnter Roll Number: ");

scanf("%d",&Roll\_num);

printf("\t\tEnter Father Name of Student: ");

fflush(stdin);

scanf("%s",&Father\_Name);

printf("\t\tEnter Subjects of Student: ");

fflush(stdin);

scanf("%s",&subj);

printf("\t\tEnter Class of student: ");

scanf("%d",&clas);

printf("\nSTUDENT %d RECORD",i);

fprintf(fptr," \n Name: %s \n Father Name=%s \n Roll Number=%d \n Subjects=%s \n Class=%d",name,Father\_Name,Roll\_num,subj,clas);

printf("\n\n\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\n\n");

}

fclose(fptr);

return 0;

}

/\* A menu driven file handling program to

Create, read, update and delete records

in a binary file \*/

#include <stdio.h>

/\* Defintion of record of type student \*/

struct student

{

int rollno;

char name[50];

int test\_score1, test\_score2, test\_score3;

float average;

char grade;

};

/\* Global data variable \*/

FILE \*file;

/\* Function prototypes \*/

char calculate\_grade(float average);

void input(struct student \*s);

void display(struct student st);

void write\_student();

void display\_all();

void display\_sp(int n);

void modify\_student();

void delete\_student();

void class\_result();

void result\_menu();

void entry\_menu();

int main()

{

int choice;

do

{

printf("\n\nMain Menu");

printf("\n1. Result Menu");

printf("\n2. Entry/Edit Menu");

printf("\n3. Exit");

printf("\nPlease select your choice (1-3): ");

scanf("%d",&choice);

switch (choice)

{

case 1:

result\_menu();

break;

case 2:

entry\_menu();

}

}while (choice != 3);

return 0;

}

void input(struct student \*s)

{

printf("\nEnter the roll number of student: ");

scanf("%d", &s->rollno);

fflush(stdin);

printf("\nEnter the Name of student: ");

gets(s->name);

printf("\nEnter the father name : ");

gets(s->fname )

s->average = (s->test\_score1 + s->test\_score2 + s->test\_score3) / 3.0;

s->grade = calculate\_grade(s->average);

}

void display(struct student st)

{

printf("\nRoll Number of student : %d", st.rollno);

printf("\nName of student : %s", st.name);

printf("\nScore in test 1 : %d", st.test\_score1);

printf("\nScore in test 2 : %d", st.test\_score2);

printf("\nScore in test 3 : %d", st.test\_score3);

printf("\nAverage score : %0.2f", st.average);

printf("\nGrade : %c", st.grade);

}

void write\_student()

{

struct student data;

file = fopen("student.dat", "ab");

printf("\n\nPlease enter the details of student \n");

input(&data);

fwrite(&data, sizeof(data), 1, file);

fclose(file);

printf("\nStudent Record Has Been Created ");

}

void display\_all()

{

struct student data;

printf("\n\nDISPLAY ALL RECORD !!!\n");

file = fopen("student.dat", "rb");

while ((fread(&data, sizeof(data), 1, file)) > 0)

{

display(data);

printf("\n====================================\n");

} fclose(file);

}

void display\_sp(int n)

{

struct student data;

int flag = 0;

file = fopen("student.dat", "rb");

while ((fread(&data, sizeof(data), 1, file)) > 0)

{

if (data.rollno == n)

{

display(data);

flag = 1;

}

}

fclose(file);

if (flag == 0)

printf("\nRecord not exist");

}

void modify\_student()

{

struct student data;

int no, found = 0;

printf("\nTo Modify ");

printf("\nPlease Enter The roll number of student: ");

scanf("%d", &no);

file = fopen("student.dat", "rb+");

while ((fread(&data, sizeof(data), 1, file)) > 0 && found == 0)

{

if (data.rollno == no)

{

display(data);

printf("\nPlease enter the new details of student \n");

input(&data);

fseek(file, - (long)sizeof(data), 1);

fwrite(&data, sizeof(data), 1, file);

printf("\n Record Updated");

found = 1;

}

}

fclose(file);

if (found == 0)

printf("\n Record Not Found ");

}

void delete\_student()

{

int no;

struct student data;

FILE \*file2;

printf("\n\nDelete Record");

printf("\nPlease Enter The roll number you want to delete: ");

scanf("%d", &no);

file = fopen("student.dat", "rb");

file2 = fopen("temp.dat", "wb");

rewind(file);

while ((fread(&data, sizeof(data), 1, file)) > 0)

{

if (data.rollno != no)

{

fwrite(&data, sizeof(data), 1, file2);

}

}

fclose(file2);

fclose(file);

remove("student.dat");

rename("temp.dat", "student.dat");

printf("\nRecord deleted.");

}

void class\_result()

{

struct student data;

file = fopen("student.dat", "rb");

if (file == NULL)

{

printf(

"ERROR!!! FILE COULD NOT BE OPEN\n\n Go To Entry Menu to create File");

printf("\n\n Program is closing ....");

return;

}

printf("\nALL STUDENTS RESULT \n");

printf("==============================================================\n");

printf("R.No.\tName\t\tTest1\tTest2\tTest3\tAverage\tGrade\n");

printf("==============================================================\n");

while ((fread(&data, sizeof(data), 1, file)) > 0)

{

printf("%-7d %-15s %-7d %-7d %-7d %-7.2f %-1c\n", data.rollno,

data.name, data.test\_score1, data.test\_score2, data.test\_score3,

data.average, data.grade);

}

fclose(file);

}

void result\_menu()

{

int rno, ans;

char choice;

printf("\n\nResult Menu");

printf("\n1. Class Result\n2. Student Report Card\n3.Back to Main Menu");

printf("\nEnter Choice (1-3)? ");

scanf("%d", &ans);

switch (ans)

{

case 1:

class\_result();

break;

case 2:

do

{

char ans;

printf("\n\nEnter roll number of student: ");

scanf("%d", &rno);

display\_sp(rno);

printf("\nDo you want to see more result (y/n)?: ");

fflush(stdin);

scanf("%c", &choice);

}while (choice == 'y' || choice == 'Y');

break;

case 3:

break;

default:

printf("\a");

}

}

void entry\_menu()

{

int choice;

printf("\n\nEntry Menu");

printf("\n1.Create Student Record");

printf("\n2.Display all students records");

printf("\n3.Search student record ");

printf("\n4.Modify student record");

printf("\n5.Delete Student record");

printf("\n6.Back to main menu");

printf("\nEnter your choice (1-6): ");

scanf("%d",&choice);

switch (choice)

{

case 1:

write\_student();

break;

case 2:

display\_all();

break;

case 3:

{

int num;

printf("\n\nPlease enter the roll number: ");

scanf("%d", &num);

display\_sp(num);

}

break;

case 4:

modify\_student();

break;

case 5:

delete\_student();

break;

case 6:

break;

default:

printf("\a");

entry\_menu();

}

}

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

THE END