(Section B)

Q. No. 2 Part (i) Differentiate:-

`void *ptr1;`	`int *ptr2;`
• This is a **void pointer** / generic pointer.	• This is a pointer pointing to a variable of **integer** data type.
• This pointer can point to **any** datatype.	• This pointer can only point to **integer** datatype.
• This cannot be dereferenced easily without **type casting**	• This can be defferenced easily by using **(*)** defrence pointer.
• Its data type is **void**	• Its datatype is the **datatype** of the variable its pointing to.
• Its used for **generic** purposes.	• Its just used to point to an **integer variable.**

• `int X; float Y;`
`void *ptr1 = &X;`
`void *ptr1 = &Y;`
	• `int *ptr2; int X = 6;`
	`ptr2 = &X;`
	`int value N;`
	`value N = *ptr2;`

Q. No. 2 Part (ii) km to meter program:-

```
#include <iostream.h>
#include <conio.h>
void main()
{
int km, m;
cout<<" Enter a distance in kilometres"<<endl;
cin >> km;
m = km*1000;
cout<<" The distance in metres is"<<m<<"metres"<<endl;
getch();
}
```

output:

Enter distance in kilometres
(5)
The distance in metres is 5000 metres

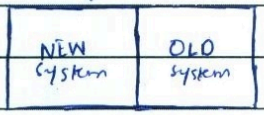
04 Answer the Question/(Part) at the space specified. 53809740

(Section B)

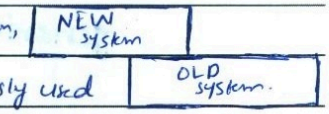
Q. No. 2 Part (iii) Implementation in SDLC/-Deployment:-

Implementation in SDLC refers to the process in which / or the set of activities to make the **system available for use.** It includes the steps of ① implementation of hardware/software ② training of personnel ③ conversion from new to old system.

⇒ **DIRECT implementation:** In direct implementation, the old system is completely discarded and the new system is implemented completely. It is the RISKIEST method of implementation, because if new system has errors, user has no means of backup.



⇒ **PARALLEL implementation:** In parallel implementation, the new and old system are simultaneously used and its the SAFEST method of all because no loss of data occurs and there is backup in form of old system.



Q. No. 2 Part (iv) PRINT SEQUENCE:-

```
#include<iostream.h>
#include<conio.h>
void main()
{
int i;
for(i=2; i<=11; i=i+3)
  {
    cout<<i<<"\t ";
  }
getch();
}
```

output:

2 5 8 11

07 Answer the Question(s) (Part) at the space specified. (Section B)

Q. No. 2 Part (ix) FILE OPENING MODES:-

① **ios::in** This file opening mode is a default mode that is used with 'ifstream' for input operations. The symbol (::) is a scope resolution operator. It is used to read something.

```
ifstream myfile;
myfile.open("myfile.txt",ios::in);
```

② **ios::out** This file opening mode is a default mode that is used with 'ofstream' for output operations. It is used to write something.

```
ofstream myfile;
myfile.open("myfile.txt",ios::out);
```

③ **ios::bin** this mode is to open a file for output operations, and for binary operations. We can read, write, and manipulate by bitwise OR (|) operator.

```
fstream myfile;
myfile.open("myfile.txt",ios::bin|ios::in|iosout);
```

05

(Section B)

Q. No. 2 Part (v) 1 1 2 3 5 8

```
int a=0 , b=1, c;
while ( a <=8 ) {
        c =a+b ;
        cout<< b <<" \t" ;
        a=b;
        b=c; }
```

Q. No. 2 Part (vi) DO-WHILE:-

```
int i=1;
do
{
 cout<<i;
 i++;
}
while (i<=10);
```

output:

12345678910

(Section B)

Q. No. 2 Part (vii) ACCESS SPECIFIERS:-

Private	Public
• Any class member in the private access specifier, cannot be accessed in main()	• Any class member written in public access specifier can be accessed in main()
• Private members cannot be accessed by derived classes.	• Public members can be accessed in derived classes.
• In private access specifier, a constructor cannot be written.	• Constructor is always in public
• It can be accessed by the friend function.	• Can also be accessed by friend function
• class num { private: ___ };	• class num { public: ___ };
• usually data members are private.	• usually member functions are private.

Q. No. 2 Part (viii)

FORMAL PARAMETER	ACTUAL PARAMETER
• formal parameters are used in function definitions and declaration	• Actual parameters are used in function calls
• values of actual parameters are copied in formal paramters.	• values of actual parameters are copied to formal ones.
• formal parameters can have only variables in brackets as parameters.	• actual paramters may ~~only~~ have variables, constants, expressions as parameters.
	• They are also called arguments.
• They are present in only the user defined function.	• They are present in the main() function.
• int func(int x) { } ← formal parameter	• void main() { func (505); } ← actual parameter

Q. No. 2 Part (x)

Pre-test	Post test
→ It is called the **while** loop.	→ It is called the **do-while** loop.
→ It is called **entry-control** loop.	→ It is called **exit-control** loop.
→ It is used to carry out the statements if and only if the condition is **true.**	→ It is used to carry out if the condition is **false**, at least ONE time.
→ It has no **semicolon**	→ it ends with a **semicolon.**
→ initialization;	→ initialization;
while (cond) { syntax	do { syntax
statements;	statements;
inc/dec;	inc/dec;
}	}
	~~while (cond);~~

Answer the Question/(Part) at the space specified.

(Section B)

Q. No. 2 Part (xi) **cin.get() for strings:**

When working with strings the function cin.get() is used with header file **<string.h>** and it is used because in this function, a **space** is considered as a **character** however in the cin statements space is considered as a **termination.** We must use **cin.ignore()** for multiple input and we can only input a string with spaces in cin.get() not any other **datatype** but cin statement allows, all datatypes as input.

```
char str[20];
cout<<"enter a string";
cin>>str;
```

```
char str[20];
cout<<"enter a string";
cin.get(str,20);
```

suppose user enter "information tech", in first one only information is displayed due to **cin limitation** but in second, complete string is displayed.

Q. No. 2 Part (xii) purpose :-

(a) `c=a*b;` it is used as an **arithematic** operator for **multiplication.**

(b) `float *p=&a;` The asterisk symbol here is used to **declare** a pointer called "p" that points to the ~~set~~ variable of type **float** and stores address of float type variable **a.** Pointer is **initialized.** in declaration statment

(c) `*p=90;` Here asterisk symbol acts as a **deference** operator and means value of some variable. The pointer p is differenced to a value 90.

09

(Section B)

Q. No. 2 Part (xiii) System Analyst :-

A system analyst is a person who has an **extensive background** in **software** and he is used in almost every phase of SDLC, due to his expertise. He arranges meetings and helps in various tasks.

- He plans the **system flow.**
- He checks the **software limitations** of a system with the help of the design phase expert and programer.
- He ensures that all **technical requirements** are met.
- He also cannibutes to the **documentation phase** by writing **user manuals.**

Q. No. 2 Part (xiv) IF-ELSE :-

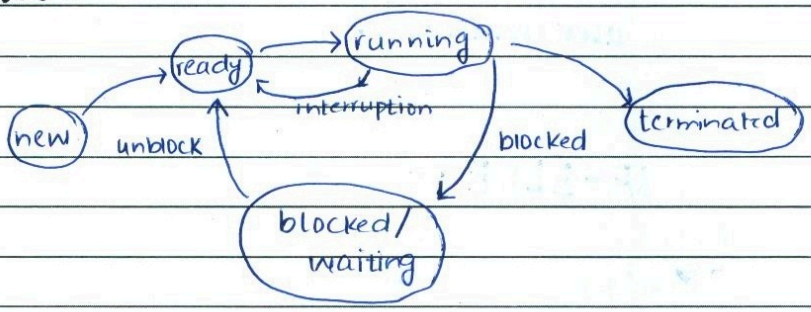
```
if (a>0) {
    s = a*a;
}
else {
    s = a*a*a;
}
```

Q. No. 3 (Page 1)

THREADS AND PROCESSES"

Process :- a process is a dynamic instance of a program or it is a program under **execution**. When a code is written it has a name 'source code' and after compilation its called 'object code', both are programs, but when a program is **loaded in memory** it becomes a **process.**

States :-



The following are the states of a process :

① New State :

It is the state when a user requests some action on a process. The process is in a 'new' state and no action has been done upon the process, but some action or activity is anticipated as requested by user.

(Section C)

Q. No. 3 (Page 2)

(2) Ready state:

In the ready state, the process is technically ready to be executed by the compiler, but it is waiting to be processed by the compiler and the resources to become free

(3) Running:

In the running state, the process is being executed and the lines of instructions with in it are being carried out by compiler.

(4) blocked / waiting:

In the blocked or waiting state, the execution of the process has been interrupted. It can either be due to the programmer who is using it for "testing" purposes or due to some real error. Due to this, the process may go back to the ready state to become ready and then it may be executed again, ~~but~~ after being unblocked.

(5) Terminated:

In the terminated state, the execution of the process is completed and terminated wholly and completely. —

Q. No. 3 (Page 3) **Features of threads :-**

(1) While a process is a program in **execution**, the thread is a **subset** of the process.
(2) The threads are **dependent** on the process, but the process is completely **independent**.
(3) The threads are operated by the **user** and the processes are operated or controlled by the operating system.
(4) Any **change** in the thread, affects all other threads as well, but a **change** in the process does not affect any other **processes**.
(5) **Multi threading** is also possible and so is **multiprocessing**.
(6) Threads have **direct access** to the data segment of a parent process, but processes have a **copy** of the data segment of parent process.
(7) Threads run in **shared memory spaces** but processes run in **separate memory spaces**.

(Section C)

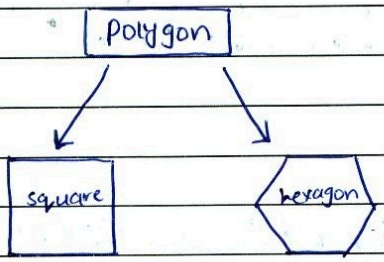
Q. No. 4 (Page 1) **INHERITANCE :-**

Inheritance is the feature of C++ object oriented programming that enables **code reusability**. In this feature we donot need to write codes again and again but instead one class **inherits** common features from another class. The class with the common features is called **'Base'** or **'parent'** class and it passes down characteristics to an inheriting class called **'sub'** or **'derived'** or **'child'** class.

Daily life examples:

Some daily life examples include that when a newborn is born, he **inherits** features like **eyes, ears, nose** from his parents.

When a new model of a car is established, it **inherits** features from previous model for example **engine, tyres** etc.



Likewise here, the sides and corners are inherited from base 'polygon' and specialised in the shapes.

Syntax :

```
class A{
    =
};
```

(Section C)

Q. No. 4 (Page 2) class B : public A {

=

};

EXAMPLE :-

```
#include<iostream.h>
class A {
        public:
            int A;
            int showA(){
                cout<<"I am base class";
            }
        };
class B : public A {
        public:
            int B;
            int showB(){
                cout<<"I am sub class";
            }
        };
void main(){
    B b1;
    b1.showA();
    b1.showB();
}
```

Here, class B is publically derived from class A.

POLYMORPHISM :-

It is a feature of C++ classes that enables us to make different functions or operators of the same name but **slightly different functionality**. Polymorphism has the following three forms,

1. **function overloading**
2. **operator overloading**
3. **virtual functions**

15

(Section C)

Q. No. 4 (Page 3) **FUNCTION OVERLOADING:** involves using a function with a singular name, but slightly different function for example it has diffrent **datatypes** or it has different **numbers** of parameters. Remember that the **return type** has nothing to do with overloading. For exaple:

```
void main()
{
  multiply(5,6);
  multiply(7,10,3);
}
```

OPERATOR OVERLOADING:- it involves using two operators for two different tasks or for different functions, for example, the **(+)** function can be used to add any two **numbers** or **concatenate** any two strings.

e.g. 20+5=25

e.g. "happy" + "life" = "happy life".

VIRTUAL FUNCTION:- it is a function whose **behavior can be overwritten** within an inheriting class using a function with same signature. For virtual function, a single property can take **many forms** and for exaple, a person can be a teacher and a mother and a daughter.

Q. No. 5 (Page 1) consonant, vowel, lower/upper case :-

```
#include <iostream.h>
#include <conio.h>
void main()
{
 char ch;
 cout<<"Enter any character (A to Z)"<<endl;
 cin>>ch;
   switch(ch){
     case-'a':
     case-'A':
     case-'e':
     case-'E':
     case-'i':
     case-'I':
     case-'O':
     case-'o':
     case-'u':
     case-'U': cout<<ch<<"is a vowel"<<endl;
               break;
     default : cout<<ch<<"is a consonant"<<endl;
               break;
   }
 cout<<"Now to check if it is an upper case or lower cake:"
     <<endl;
 if(ch>='a'&&ch<='z')
     {
       cout<<"the character is in lower case"<<endl;
     }
```

(Section C)

Q. No. 5 (Page 2)

```
else if( ch >='A' && ch <='Z')
        {
        cout<<"the character is in upper case"<<endl;
        }
else
        {
        cout<<"invalid output";
        }
getche();
}
```

output

enter any character(CAPS)

: A

A is a vowel

A is in uppercase

(Section C)

Q. No. 5 (Page 3)

(Section C)

Q. No. 6 (Page 1) ARRAYS:-

```
#include <iostream.h>
#include <conio.h>
void main() {
    int arr[5], sum=0, avg=0;
    cout<<" enter 5 numbers with space"<<endl;
    for (i=0; i<5; i++)
      {
        cin>>arr[i];
        sum = sum + arr[i];
      }
    cout<<" NOW to display in reverse order"<<endl;
    for (i=4; i>=0; i--)
      {
        cout<<arr[i]<<"\t"<<endl;
      }
    avg = sum/5;
    cout<<" the average is "<<avg;
    getch();
}
```

Output:

```
enter 5 numbers with space
1 2 3 4 5
NOW to display in reverse order
5 4 3 2 1
the average is 3
```

Q. No. 6 (Page 2)

(Section C)

Q. No. 6 (Page 3)

Space for rough work

23

Space for rough work

Space for rough work

for(a=1; a<=10; a++)

1
2
3
4
5
:
10

for(a=1; a<=10; a=a+3)

1
4
7
10

for(a=10; a>1; a--)

10
9
8
7
6
5
4
3
2

C = 0+1
C = 1

1

a = 1
b = 1

C = 1+1

for(a=1; a<10; a++)

int a=0; b=1; c;
while(a<=8){
c=a+b;
cout<<b;
a=b;
b=c;}

c = 1,

a = 1,
b = 1,

1 2

1 1 2 3 5 8

c = 1+1
cout<<b.

a = 1;
b = 2;

c = 1+2 = 3;
a = 2;
b = 3;

c = 3+2;

c = 5;
a = 3,
b = 5;

a = 5;
c = 5+3,
a = 5,
b = 8;

c = 8+5;
b = 8;
a = 8;