(Section B)

Q. No. 2 Part (i)

`void *ptr1;`	`int *ptr2;`
Name of pointer	
Pointer is named ptr1	Pointer is named ptr2
Data type	
void is datatype. This pointer is also called ~~generic pointer~~.	int is the data type
Purpose	
It points to a variable of any data type (int, char, float etc)	It points to a variable of only integer data type. It can not point to variable of char, float or any other data type

`char b;`
`int a;`
`ptr1 = &a;` // valid
`ptr2 = &b;` // valid
	`char b;`
	`int a;`
	`ptr2 = &b;` // invalid
	`ptr2 = &a;` // valid

Q. No. 2 Part (ii) **Distance in Meters**

```
#include <iostream.h>
#include <conio.h>
void main()
{
    int m, km;
    cout<< "Enter distance in kilometers:";
    cin>> km;
    m= 1000 * km;
    cout<< "\nDistance in meters=" << m;
    getch();
}
```

Output:

```
Enter distance in kilometers: 2
Distance in meters= 2000
```

04 Answer the Question(Part) at the space specified. 53809739

(Section B)

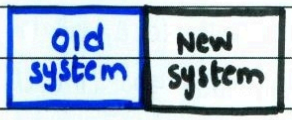
Q. No. 2 Part (iii) **Implementation in SDLC**

Implementation is an important phase in SDLC. It is also called **deployment phase.** This phase includes set of activities that make a system **available for use.** Its activities include

- Installing and activating hardware and software
- Conversion of old system to new system
- Training users or computer operation personnel for its use.

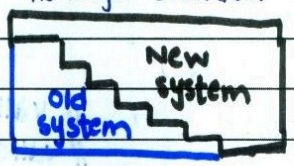
Methods:

(1) Direct Implementation:



In direct implementation, old system is dropped **completely** and new system is implemented at the same time. Old system is no longer available.

(2) Phased Implementation:



In phased implementation, the new system is **gradually** introduced and old system is **progressively** discarded.

Q. No. 2 Part (iv)

2 5 8 11

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

(Section B)

Q. No. 2 Part (ix) File Opening Modes

If we want to open a file for a **specific purpose** we use mode as a second argument in open() function.

open (filename, mode);

ios::in ios::in mode is used to open a file for input (reading purposes)

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

ios::out ios::out mode is used to open a file for output (writing purposes) in file.

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

ios::binary ios::binary mode opens the file in binary mode

```
myfile.open ("sales.bin", ios::binary)
```

Q. No. 2 Part (v)

Corrected Code:

```
int a[10];          // semicolon is statement terminator
a[9] = 30;          /* index is always an integer value.
                       index starts from 0 so the highest index
                       of this array will be 9 */
a[0] = 20;          // index is never negative (starts from 0)
```

Q. No. 2 Part (vi) **do-while Loop**

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

(Section B)

Q. No. 2 Part (vii)

Private Access Specifier	Public Access Specifier
Access within Class, Friend Function	
They are accessible within their class	They are accessible within class
Usage	
Data members are usually private	Member functions are mostly public
Access in Derived Classes	
They are not accessible in derived classes	They are accessible in derived classes
Access in main()	
They are not accessible in main()	They are accessible in main()
Default / Not Default	
They are default access specifiers	It is not a default access specifier

---

Q. No. 2 Part (viii)

Constructor	Destructor
Execution and Purpose	
Constructors are executed automatically when an object is **created** and they are usually used for initialization. They **allocate** memory for the object	Destructors are executed when an object is **destroyed**. It **de-allocates** the memory reserved for the object.
Symbol	
It has **no** symbol. e.g. const() {}	It has **tilde** (~) symbol e.g. ~const() {}
Parameters / Overloading	
Constructors take parameters so they can be overloaded by using different number or types of parameters.	Destructors do not take arguments so they can not be overloaded.

Q. No. 2 Part (x)

Arrays

(a) Decleration

```
float arr [3][3];
```

(b) Store 10.5

```
arr [0][0] = 10.5;
```

(c) Display last element

```
cout << arr [2][2];
```

(Section B)

Q. No. 2 Part (xi)

=	==
Type of Operator	
It is an assignment operator	It is a relational operator
Checking for Equality	
It is not used to check for equality	It is used to check for equality
Purpose	
It is used to assign value of a variable/constant/result of expression to a variable.	It is used to check if both sides of the operator are equal.
Order of Precedence	
It has lower order of precedence	It has higher order of precedence
Example	
`int a=2, b=3` [variable = expression] value 2 is assigned to a, 3 to b	`if (b==2) cout << b;` it checks if b is equal to 2 or not

Q. No. 2 Part (xii) **Purpose of Asterisk (*)**

a. c=a*b;

Here asterisk is used for **multiplication.** a is multiplied with b and result is stored in c.

b. float *p=&a;

Here asterisk is used for **pointer decleration.** A pointer of name p is declared which will point to variable of type float.

c. *p=90;

Asterisk is used for **dereferencing** a pointer.
90 is stored in the variable whose address is pointed by the pointer.

(Section B)

Q. No. 2 Part (xiii) **Responsibilities of System Analyst**

System analyst is a personnel involved in **SDLC** who is a professional in **software development.** He studies the problem, plans solution, recommends system and coordinates development. He has 3 main roles:

(1) Requirement Documents: A system analyst **interacts** with customers to know the requirements and **documents** them in form of business requirement documents. He defines **technical requirements** and contributes to **user manuals.**

(2) Managing Testing: A system analyst manages system **testing** and helps programmers during **system development**

(3) Software Limitations: System analysts plans a system flow and interacts with **designers** to know software limitations.

Q. No. 2 Part (xiv) **if-else statement**

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

(Section C)

Q. No. 3 (Page 1)

Process

Process is a program in **execution**. When we write a code, and it is compiled, an object code is created. Source code and object code both are programs but when program is executed ~~and~~ it becomes a process.

Program is an executable code stored as text in hard disk.
Process is a dynamic instance of a program that is being executed. It uses computer resources like CPU time, memory input output devices.

States of Process

A process has 5 states:

(1) New State:

New state is the **first** state of a process when the process is **created.**

(2) Ready State:

Ready state is the second state of a process when a process is **ready** to be executed but it is waiting to be **assigned** to a CPU by **operating system.**

(3) Running State:

Running state is the third state when a process is being **executed** by **CPU**. It has been assigned to a CPU by the operating system.

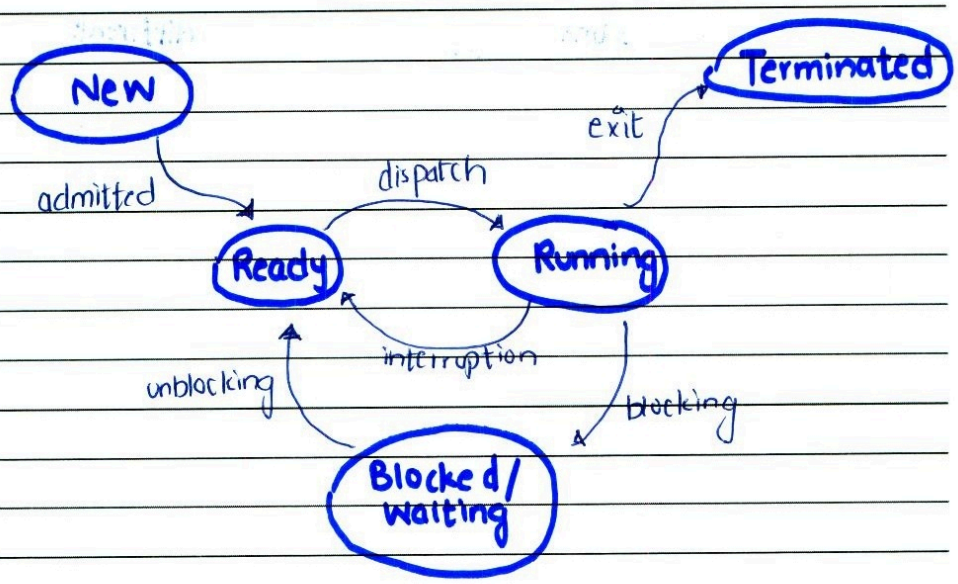
Q. No. 3 (Page 2)

(4) Blocked / Waiting State:

Process is in blocked state when it is not being executed. It is waiting for a **resource** to become **available**.

(5) Terminated State:

A Process is in terminated state when it has **completed** its execution.



Thread	Process
(1) Thread is a **subset** of a process	(1) Process is a dynamic instance of a program in execution by CPU
(2) It has **direct** access to data segment of its process	(2) It has its own copy of the data segment of its parent process.

(Section C)

Q. No. 3 (Page 3)

Thread	Process
(3) Execution of threads is controlled by **programmer**	(3) Execution of process is controlled by **operating system**
(4) They are **dependent** and change in one thread may effect behavior of other threads.	(4) They are **independent** and change in one process does not affect other process
(5) Threads run in the **same** memory space	(5) Processes run in **different** memory locations.

(Section C)

Q. No. 4 (Page 1)

FEATURES OF OBJECT ORIENTED PROGRAMMING

(1) Code Reusability : Inheritance

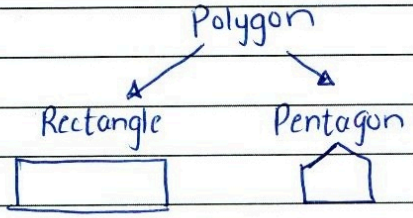
Inheritance is a feature of object oriented programming that enables **code reusability.** This feature allows classes to be created from an **existing class.**

Base class: Base class is the class from which other classes are derived. It is also called **parent** class.

Derived class: Derived class is the class which is derived from a base class. It is also called **child or subclass.**

Examples:

(1) Consider a base class polygon. It has a number of base classes like pentagon and rectangle. They have common features like each of them can be described by means of two sides and two angles.



(2) Consider base class Patient. It has two derived classes indoor patient and outdoor patient. Common characteristics are age, gender, name.
Indoor patient has characteristics bed no, ward no.
Outdoor patient has characteristic next date of visit.

(Section C)

Q. No. 4 (Page 2)

Syntax:

```
class A                          // base class
{
};
class B: public A                // derived class
{
};
```

Example:

```
class A
{ public:
    void showA()
    { cout<< "I am base class A";
    }
};
class B: public A
{ public:
    void showB()
    { cout<< "I am derived class B";
    }
};
void main()
{ B b1;
  b1.showA();
  b1.showB();
}
```

Q. No. 4 (Page 3)

(2) Multiple ways of Functionality: Polymorphism

Polymorphism comes from word **poly** which means multiple functionalities of operators or functions. Polymorphism provides feature to use operator or function in different ways.

(a) Virtual Functions:

Virtual functions are functions whose behavior can be **overridden** within an inheriting class. by using a function having the **same signature.**

(b) Operator Overloading:

A singe operator can be used in **multiple ways.** For example '+' operator can be used for addition of two integers, floating points or concatenation of two strings

```
7 + 5
2.5 + 2.5
"federal" + "Board"
```

(c) Function Overloading

Functions can be overloaded. We can use functions having the same name as long as they have different **number of type of parameters.** Compiler is able to **disambiguate** them on this basis and the function is executed whose parameters **match** the arguments passed during function call.

```
int Sum (int x, int y)
{
    return (x+y);
}
int Sum (int x, int y, int z)
{
    return (x+y+z);
}
```

(Section C)

Q. No. 5 (Page 1)

Alphabet Program

```
#include<iostream.h>
#include <conio.h>
void main()
{
    char ch;
    cout<< " Enter an alphabet: ";
    cin>>ch;
    switch (ch)                    // part a: vowel or consonant
    {
      case 'a':
      case 'e':
      case 'i':
      case 'O':
      case 'u':
      case 'A':
      case 'E':
      case 'I':
      case 'O':
      case 'U': cout << "\n It is a vowel";
                break;
      default : cout << "\nIt is a consonant";
                break;
    }
// part b: lowercase or uppercase
if ((ch>='a') && (ch<='z'))
cout<<"\n"<< ch << " is a lowercase letter";
```

Q. No. 5 (Page 2)

```
else if ((ch>='A') &&(ch<='Z'))
        cout<< "\n" << ch << " is an uppercase letter";
else
        cout<< "Invalid input";
getch();
}
```

Output:

Enter an alphabet: a
It is a vowel
It is a lowercase letter

(Section C)

Q. No. 5 (Page 3)

(Section C)

Q. No. 6 (Page 1)

```
#include <iostream.h>
#include <conio.h>
void main()
{
    int arr[5];
    int i, sum=0;
    float avg ;
    cout<<"Enter 5 numbers";
    for (i=0; i<=4; i++)
    {   cin>>arr[i];
        sum= sum+ arr[i];
    }
    avg= sum/5;
    cout<<"\nNumbers in reverse order:";
    for (i=4; i>=0; i--)
    {
        cout<<arr[i]<<" ";
    }
    cout<<"\nAverage ="<<avg;
    getch();
}
```

Output:

Enter 5 numbers: 1 2 3 4 5
Numbers in reverse order: 5 4 3 2 1
Average = 3.0

(Section C)

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