Department of Computer Science & Engineering
Course Name: Object-Oriented Programming in C++
Course Code: CSE-1221; Even Semester.

Object-Oriented Programming (OOP)

⇒ **Procedure Oriented Programming (POP):** Conventional programming using high level languages such as COBOL, FORTRAN and C is commonly known as procedure-oriented programming (POP). **Characteristics of POP:**

1. Emphasis is on doing things (algorithms).
2. Large programs are divided into smaller programs known as functions.
3. Most of the functions share global data.
4. Data move openly around the system from function to function.
5. Functions transform data from one form to another.
6. Employs top down approach in program design.

⇒ **Object-Oriented Programming (OOP):** It was invented to overcome the drawbacks of the procedure-oriented programming. It employs the bottom-up programming approach.

Characteristics of OOP:

1. Emphasis is on data rather than procedure.
2. Programs are divided into what are known as objects.
3. Data structures are designed such that they characterize the objects.
4. Functions that operate on the data of an object are tied together in the data structure.
5. Data is hidden and can not be accessed by external functions.
6. Objects may communicate with each other through functions.
7. New data and functions can be easily added whenever necessary.
8. Follows bottom-up approach in program design.

⇒ **Benefits of OOP:** The principal advantages / benefits are:

1. We can eliminate redundant code and extend the use of existing classes.
2. To build secure programs.
3. It is possible to have multiple instances of an object to co-exist without any inheritance.
4. It is possible to map objects in the problem domain to those in the program.
5. It is easy to partition the work in a project based on objects.
6. It can be easily upgraded from small to large systems.
7. Software complexity can be easily managed.

⇒ **Application of OOP:** The promising areas for applications of OOP include:

1. Real-time systems,
2. Simulation and modeling, → Qarc
3. Object-oriented databases,
4. Hypertext, hypermedia and expertext,
5. Artificial Intelligence and expert systems,
6. Neural networks and parallel programming,
7. Decision support and office automation systems,
8. CIM / CAM / CAD systems.

Example:

Data Abstraction 1. TV, which you can turn on and off change the channel, adjust the volume, and add. external components such as speaker, VCR's, and DVD players, Bt you do not know its internal details, that is you do not know how it receives singhals over the air on through a cable, how it translates them and finally displays them on the screen.

→ we can say a TV clearly seperates it internal implementation from its external interface and you can play with its interfaces like power button, channel changer, volume control, without having zero knowledge of its internals.

Advantages; (i) Helps the user to avoid writing the low level code.
(ii) Avoid Code duplication & increase reusability
(iii) can change internal implementation of class without affecting user
(iv) Helps to increase security of program.

Same as: if we talk in terms of C++ Programming c++ classes provides great level of data abs...

They provide sufficient public methods to the outside world to play with functionality of the object and to manipulate object data, state without actually knowing how class has been implemented internally.

Example:

```
#include<iostream>
using namespace std;
int main()
{
  cout << "Hello world in c++" << endl;
} return 0;
```

[illegible], cout [illegible]. Hello World C++ disp[illegible] [illegible], Bt [illegible] [illegible] [illegible] [illegible] [illegible] [illegible] library [illegible]

Basic Concept / Properties of OOP:

It is necessary to understand some of the concepts used extensively in object-oriented programming. These includes:

1. Objects,
2. Classes,
3. Data abstraction and encapsulation,
4. Inheritance,
5. Polymorphism,
6. Dynamic binding,
7. Message passing.

Objects: Object is the instance of a class. A class provides a blueprint for objects. So we can create an object from a class. The objects of a class are declared with the same sort of declaration that we declare variables of basic types.

Example:

```
class person
{
   char name[20];
   int id;
public:
   void getdetails(){}
};

int main()
{
  person p1; //p1 is a object
}
```

Object take up space in memory and have an associated address like a record in pascal or structure or union in C. When a program is executed the objects interact by sending messages to one another. Each object contains data and code to manipulate the data. Objects can interact without having to know details of each others data or code, it is sufficient to know the type of message accepted and type of response returned by the objects.

Class: Class is a user-defined data type. Class is a blueprint of data and functions or methods. Class does not take any space.

Syntax for class:

```
class class_name
{
 private:
   //data members and member functions declarations
 public:
   //data members and member functions declarations
 protected:
   //data members and member functions declarations
};
```

Class is a user defined data type like structures and unions in C.

⇒ **Encapsulation:** is an object oriented programming concept that binds together the data and functions (method) that manipulate the data and that keeps both safe from outside interference (ಹಸ್ತಕ್ಷೇಪ) and misuse (ದುರ್ಬಳಕೆ).

Example:

```
class Box
{
public:
    double getVolume(void)
    {
        return length * breadth * height;
    }
private:
    double length;   // Length of a Box.
    double breadth;  // breadth of a
    double height;   // height of a
};
```

Here, variables length, breadth, height are private. That means they can be accessed only by other members of the box class, and not by any other part of your program.

By default class variables are private but in case of structure it is public. in above example person is a class.

Encapsulation: Wrapping up(combing) of data and functions into a single unit is known as encapsulation. The data is not accessible to the outside world and only those functions which are wrapping in the class can access it. This insulation of the data from direct access by the program is called data hiding or information hiding. In C++, it is achieved using the access specifiers i.e. public, private and protected .

Data Abstraction: Data abstraction refers to, providing only needed information to the outside world and hiding implementation details. For example, consider a class Complex with public functions as getReal() and getImag(). We may implement the class as an array of size 2 or as two variables. The advantage of abstractions is, we can change implementation at any point, users of Complex class wont't be affected as out method interface remains same. Had our implementation be public, we would not have been able to change it.

Inheritance: Inheritance is used to inherit the property of one class into another class. It facilitates you to define one class in term of another class.

Polymorphism: Polymorphism means one name, multiple forms. It allows us to have more than one function with the same name in a program. It also allows us overloading of operators so that an operation can exhibit different behaviors in different instances.

Late Binding | **Dynamic Binding:** → is when memory is allocated at run time, by means of the new operator. In dynamic binding, the code to be executed in response to function call is decided at runtime. C++ has virtual functions to support this.. Example: double *p; p = new double[100];

Message Passing: Objects communicate with one another by sending and receiving information to each other. A message for an object is a request for execution of a procedure and therefore will invoke a function in the receiving object that generates the desired results. Message passing involves specifying the name of the object, the name of the function and the information to be sent.

Data Binding:

Early Binding | Satic Binding: Static binding is when memory is allocated at compile time as with the array declaration. Example: double a[400];

⇒ **Distinguish Classes from Objects:**
Classes are used to specify the structure of the data. They define data type. You can create any number of objects from a class. Objects are the instances of classes.

⇒ **Operator Overloading:** Operator overloading is the process of making an operator to exhibit different behaviors in different instances is known as operator overloading.

⇒ **Function Overloading:** Function overloading is using a single function name to perform different types of tasks.

polymorphism is extensively used in implementing inheritance.

Polymorphism: occurs when there is a hierarchy of classes and they are related by inheritance.

→ It means that a call to a member function will cause a different function to be executed depending on the type of object that invokes (or) the function (method).

Why use: Eliminate errors, code smells (bad), refactor ( ), and navigate (into your code), comply with coding standards & more.

Difference between C and C++ :

C	C++
1. procedural programming L	1. OOP —
2. Does not support	2. It support object, class
3. Does not support.	3. It support polymorphism, encapsulation, inheritance.
4. Does not	4. supports Program & function overloading
5. Does not allows.	5. functions can be used inside a structure.
6. scanf() and Printf()	6. Cin and Cout
7. support malloc, calloc(), free (dynamic M.A)	7. new, delete operator. Mallocation ↓ de-allocation

[illegible], | private, Public, Protected [illegible] → Access specifier
Data members → Variables to be used.
Member function → methods to access data mem

Object-Oriented Languages: There are two types of object-oriented language which are given below:

1. Object-based programming language and
2. Object-oriented programming language.

Object-based programming language: Object-based programming is the style of programming that primarily supports encapsulation and object identity. Major features that are required for object-based programming are:

1. Data encapsulation,
2. Data hiding and access mechanism,
3. Automatic initialization and clear-up of objects,
4. Operator overloading.

Object-oriented programming language: Object-oriented programming incorporates all of object-based programming features along with two additional features namely: Inheritance and dynamic binding i.e:

object-based features + inheritance + dynamic binding.

What is C++ ?

Answer: C++ is an object oriented programming language created by Bjarne Stroustrup. It is released in 1985.

What are the advantages of C++ ?

Answer: C++ doesn't only maintains all aspects from C language, it also simplify memory management and add several features like:

- Includes a new data type known as a class.
- Allows object oriented programming.

Structure Vs class in C++:

1. Members of a class are private by default and members of struct are public by default. For example program 1 fails in compilation and program 2 works fine.

```
// Program 1
#include <stdio.h>

class Test {
   int x; // x is private
};
int main()
{
 Test t;
 t.x = 20; // compiler error because x is private
 getchar();
 return 0;
}
```

```
//Program 2
#include<stdio.h>
struct Test
{
   int x; // x is public.
};
int main()
{
  Test t;
  t.x = 20; // works fine bcoz x is public
  getchar();
  return 0;
}
```

⇒ **Accessing Data Members & Member functions:** The data members and member functions of a class can be accessed using the (.) operator with the object.

Example:

```
#include<iostream>
using namespace std;
class Geeks
{
    public:     // Access specifier.
    string geekname;  // Data member.
    void printname()  // member function
    {
        cout<<"Geekname is:"<<geekname;
    }
};
int main()
{
    Geeks obj1;  // object declare
    obj1.geekname = "Abhi"; // Accessing data member
    obj1.printname(); // Accessing member function
    return 0;
}
```

Output: Geekname is: Abhi

Parameter / Argument

2. When deriving a struct from a class/struct, default access-specifier for a base class/struct is public. And when deriving a class, default access specifier is private.

Difference between C structures and C++ structures:

In C++, struct and class are exactly the same things, except for that struct defaults to public visibility and class defaults to private visibility.

Some important differences between the C and C++ structures:

1. **Member functions inside structure**: Structures in C cannot have member functions inside structure but Structures in C++ can have member functions along with data members.
2. **Direct Initialization:** We cannot directly initialize structure data members in C but we can do it in C++.

```
// C program to demonstrate that direct
// member initialization is not possible in C
#include<stdio.h>
 struct Record
{
  int x = 7;
};
 // Driver Program
int main()
{
   struct Record s;
   printf("%d", s.x);
   return 0;
}
/* Output :  Compiler Error
  6:8: error: expected ':', ',', ';', '}' or
 '__attribute__' before '=' token
 int x = 7;
    ^
 In function 'main': */
```

Output:

```
7
```

- **Using struct keyword:** In C, we need to use struct to declare a struct variable. In C++, struct is not necessary. For example, let there be a structure for Record. In C, we must use "struct Record" for Record variables. In C++, we need not use struct and using 'Record' only would work.
- **Static Members:** C structures cannot have static members but is allowed in C++.

Member function of classes:

⇒ There are two ways to define a member function: (i) inside class definition
(ii) outside class definition.

N.B: To define a member function outside the class definition we have to use the scope resolution (::) operator along with class name and function name.

Example:

```
#include <iostream>
using namespace std;
class Geeks
{
  public:
    string geekname;
     int id;
     void printname();   // is not define outside the class.
     void printid()
     {
       cout<< "Geek id is: "<< id;
     }
};
void Geeks:: printname()
  {
    cout << " Geekname is: " << geekname;
  }
int main() {
     Geeks obj1;
     obj1.geekname = "xyz";
     obj1.id  =15;
     obj1.printname(); // call printname()
     obj1.printid();   // call printid
} return 0;
```

output:
Geek id is: 15
Geek name is: xyz

```
// C program with structure static member
struct Record
{
   static int x;
};
 // Driver program
int main()
{
   return 0;
}
/* 6:5: error: expected specifier-qualifier-list
   before 'static'
    static int x;
    ^*/
```

□

This will generate an error in C but no error in C++.

□ **sizeof operator:** This operator will generate **0** for an empty structure in C whereas **1** for an empty structure in C++.

```
// C program to illustrate empty structure
#include<stdio.h>
 //empty structure
struct Record
{
};
 //Driver program
int main()
{
   struct Record s;
   printf("%d\n",sizeof(s));
   return 0;
}
```

□ Output in C:

0

Output in C++:

1

□ **Data Hiding:** C structures does not allow concept of Data hiding but is permitted in C++ as C++ is an object oriented language whereas C is not.

□ **Access Modifiers:** C structures does not have access modifiers as these modifiers are not suppoted by the language. C++ structures can have this concept as it is inbuilt in the language.

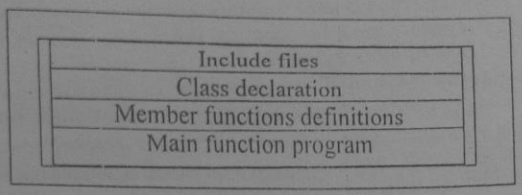
Nested classes in C++ : A nested class is a class which is declared in another enclosing class. A nested class is a member and as such has the same access rights as any other member.

The members of an enclosing class have no special access to members of a nested class, the usual access rules should be obeyed.

```
class enclosing
{
    int x;
    class nested.  //nested class declaration
    {
        int y;
        void nestedfun(enclosing *e)
        {
            cout << e-> x;
        }
    };
};
int main()
{
}
```

Discuss the basic structure of C++ program.

Answer: A typical C++ program would contain four sections which is shown as figure (a). These sections may be placed in separate code files and then compiled independently or jointly.



The class declarations are placed in a header file and the definitions of member functions go into another file. This approach enables the programmer to separate the abstract specification of the interface (class definition) from the implementation details (member functions definitions). Finally, the main program that uses the class is placed in a third file which "includes" the previous two files as well as any other files required.

C++ Data Types:

All variables use data-type during declaration to restrict the type of data to be stored. Therefore, we can say that data types are used to tell the variables the type of data it can store. Whenever a variable is defined in C++, the compiler allocates some memory for that variable based on the data-type with which it is declared. Every data type requires different amount of memory.

Data types in C++ is mainly divided into two types:

1. **Primitive Data Types**: These data types are built-in or predefined data types and can be used directly by the user to declare variables. example: int, char , float, bool etc. Primitive data types available in C++ are:
    - Integer
    - Character
    - Boolean
    - Floating Point
    - Double Floating Point
    - Valueless or Void
    - Wide Character
2. **Abstract or user defined data type**: These data types are defined by user itself. Like, defining a class in C++ or a structure.

This article discusses **primitive data types** available in C++.

- **Integer**: Keyword used for integer data types is **int**. Integers typically requires 4 bytes of memory space and ranges from -2147483648 to 2147483647.
- **Character**: Character data type is used for storing characters. Keyword used for character data type is **char**. Characters typically requires 1 byte of memory space and ranges from -128 to 127 or 0 to 255.
- **Boolean**: Boolean data type is used for storing boolean or logical values. A boolean variable can store either *true* or *false*. Keyword used for boolean data type is **bool**.

- **Floating Point**: Floating Point data type is used for storing single precision floating point values or decimal values. Keyword used for floating point data type is **float**. Float variables typically requires 4 byte of memory space.
- **Double Floating Point**: Double Floating Point data type is used for storing double precision floating point values or decimal values. Keyword used for double floating point data type is **double**. Double variables typically requires 8 byte of memory space.
- **void**: Void means without any value. void datatype represents a valueless entity. Void data type is used for those function which does not returns a value.
- **Wide Character**: Wide character data type is also a character data type but this data type has size greater than the normal 8-bit data type. Represented by **wchar_t**. It is generally 2 or 4 bytes long.

Data type Modifiers: As the name implies, data type modifiers are used with the built-in data types to modify the length of data that a particular data type can hold. Data type modifiers available in C++ are:

- **Signed**
- **Unsigned**
- **Short**
- **Long**

Data Type	Size(in bytes)	Range
short int	2	-32,768 to 32,767
unsigned short int	2	0 to 65,535
unsigned int	4	0 to 4,294,967,295
int	4	-2,147,483,648 to 2,147,483,647
long int	4	-2,147,483,648 to 2,147,483,647
unsigned long int	4	0 to 4,294,967,295
long long int	8	-(2^63) to (2^63)-1
unsigned long long int	8	0 to 18,446,744,073,709,551,615
signed char	1	-128 to 127
unsigned char	1	0 to 255
float	4	
double	8	
long double	12	
wchar_t	2 or 4	1 wide character

Below table summarizes the modified size and range of built-in data types when combined with the type modifiers: **Note** : Above values may vary from compiler to compiler. In above example, we have considered GCC 64 bit. We can display the size of all the data types by using the size_of() function and passing the keyword of the data type as argument to this function as shown below:

```
/// C++ program to sizes of data types:
#include<iostream>
using namespace std;
 int main()
{
   cout << "Size of char : " << sizeof(char) << " byte" << endl;
   cout << "Size of int : " << sizeof(int) << " bytes" << endl;
   cout << "Size of short int : " << sizeof(short int) << " bytes" << endl;
   cout << "Size of long int : " << sizeof(long int) << " bytes" << endl;
   cout << "Size of signed long int : " << sizeof(signed long int) << " bytes" << endl;
   cout << "Size of unsigned long int : " << sizeof(unsigned int) << " bytes" << endl;
   cout << "Size of float : " << sizeof(float) << " bytes" <<endl;
   cout << "Size of double : " << sizeof(double) << " bytes" << endl;
   cout << "Size of wchar_t : " << sizeof(wchar_t) << " bytes" <<endl;

   return 0;
}
```

Output:

```
Size of char : 1 byte
Size of int : 4 bytes
Size of short int : 2 bytes
Size of long int : 8 bytes
Size of signed long int : 8 bytes
Size of unsigned long int : 4 bytes
Size of float : 4 bytes
Size of double : 8 bytes
Size of wchar_t : 4 bytes
```

⇒

Operators in C / C++:

Operators are the foundation of any programming language. Thus the functionality of C/C++ programming language is incomplete without the use of operators. We can define operators as symbols that helps us to perform specific mathematical and logical computations on operands. In other words we can say that an operator operates the operands. For example, consider the below statement:

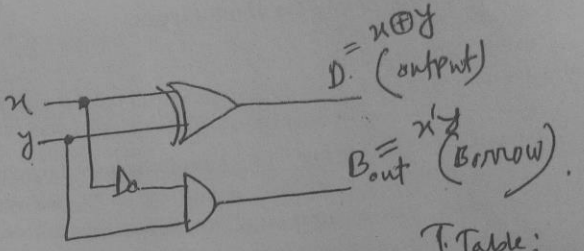
```
c = a + b;
```

Here, '+' is the operator known as *addition operator* and 'a' and 'b' are operands. The addition operator tells the compiler to add both of the operands 'a' and 'b'. C/C++ has many built-in operator types and they can be classified as:

- **Arithmetic Operators**: These are the operators used to perform arithmetic/mathematical operations on operands. Examples: (+, -, *, /, %,++,–).
  Arithmetic operator are of two types:

Subtractor: used for subtraction.

① Half Subtractor: 2-bit:



$D = x \oplus y$ (output)

$B_{out} = x'y$ (Borrow)

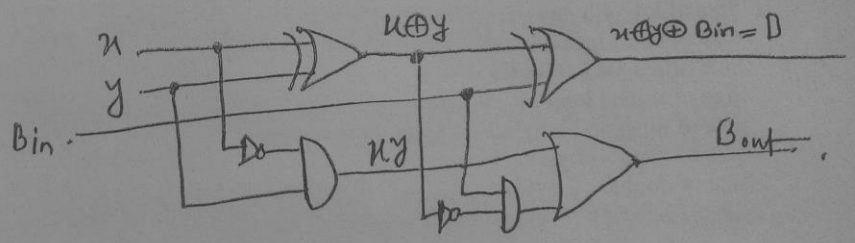
T. Table:

A	B	D	Bout
0	0	0	0
0	1	1	1
1	0	1	0
1	1	0	0

② Full Sub: 3-bit subtraction.

T. Table:

A	B	Bin	D	Bout
0	0	0	0	0
0	0	1	1	1
0	1	0	1	1
0	1	1	0	1
1	0	0	1	0
1	0	1	0	0
1	1	0	0	0
1	1	1	1	1



$D = x \oplus y \oplus B_{in}$

$B_{out} = x'B_{in} + x'y + yB_{in}$

1. **Unary Operators:** Operators that operates or works with a single operand are unary operators.
   For example: (++ , –)
2. **Binary Operators:** Operators that operates or works with two operands are binary operators. For example: (+ , – , * , /)

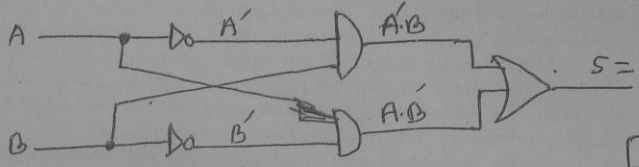
- **Relational Operators:** Relational operators are used for comparison of the values of two operands. For example: checking if one operand is equal to the other operand or not, an operand is greater than the other operand or not etc. Some of the relational operators are (==, > , = , <=). To learn about each of these operators in details go to this link.
- **Logical Operators:** Logical Operators are used to combine two or more conditions/constraints or to complement the evaluation of the original condition in consideration. The result of the operation of a logical operator is a boolean value either true or false. To learn about different logical operators in details please visit this link.
- **Bitwise Operators:** The Bitwise operators is used to perform bit-level operations on the operands. The operators are first converted to bit-level and then calculation is performed on the operands. The mathematical operations such as addition , subtraction , multiplication etc. can be performed at bit-level for faster processing. To learn bitwise operators in details, visit this link.
- **Assignment Operators:** Assignment operators are used to assign value to a variable. The left side operand of the assignment operator is a variable and right side operand of the assignment operator is a value. The value on the right side must be of the same data-type of variable on the left side otherwise the compiler will raise an error. Different types of assignment operators are shown below:
  - "=": This is the simplest assignment operator. This operator is used to assign the value on the right to the variable on the left.
    For example:
  - a = 10;
  - b = 20;
  - ch = 'y';
  - "+=":This operator is combination of '+' and '=' operators. This operator first adds the current value of the variable on left to the value on right and then assigns the result to the variable on the left.
    Example:
  - (a += b) can be written as (a = a + b)
    If initially value stored in a is 5. Then (a += 6) = 11.
  - "-=":This operator is combination of '-' and '=' operators. This operator first subtracts the current value of the variable on left from the value on right and then assigns the result to the variable on the left.
    Example:
  - (a -= b) can be written as (a = a - b)
    If initially value stored in a is 8. Then (a -= 6) = 2.
  - "*=":This operator is combination of '*' and '=' operators. This operator first multiplies the current value of the variable on left to the value on right and then assigns the result to the variable on the left.
    Example:

Adder: is a digital circuit that performs addition of numbers.

Uses: (i) computer processors.
(ii) used to calculate addresses.
(iii) used to table indices,
(iv) increment & decrement operators.

Type: 2 types :—

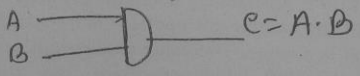
(a) **Half Adder:** For 2-bit addition.



Truth Table:

A	B	C	S
0	0	0	0
0	1	0	1
1	0	0	1
1	1	1	0

Fig(a): Sum



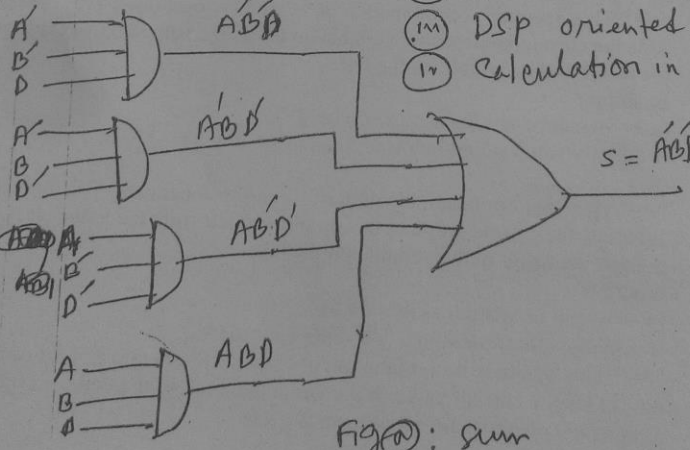
Fig(b): Carry

Use:
(i) Processor
(ii) ALU

(b) **Full Adder:** 3-bit Addition.

Advantages:
(i) Reduce circuit Complexity.
(ii) At networking side the F.A. is used
(iii) DSP oriented system
(iv) Calculation in Addition & subtraction

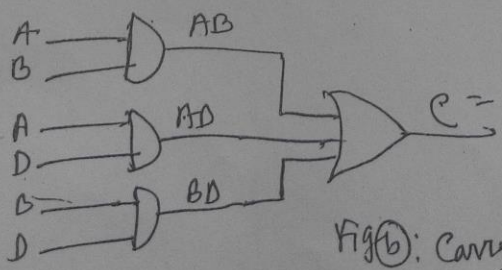
(i) Processor chip
(ii) ALU.



Truth Table:

A	B	C	C	S
0	0	0	0	0
0	0	1	0	1
0	1	0	0	1
0	1	1	1	0
1	0	0	0	1
1	0	1	1	0
1	1	0	1	0
1	1	1	1	1

Fig(a): Sum



Fig(b): Carry