Unit-2

Basics of C++ programming

Date. _____
Page No. _____

C++ is an object-oriented programming language. It was developed by Bjarne Stroustrup at AT&T Bell laboratories in early 1980's. Stroustrup wanted to combine the best of both languages and create a more powerful language that could support object-oriented programming features and still retain the power and elegance of C. The result was C++. The idea of C++ comes from the C increament operater ++, thereby suggesting that C++ is an augmented (incremented) version of C.

(Application) → C++ is a ~~very~~ versatile language for handling very large programs. It is suitable for virtually any programming task including development of editors, compilers, databases, communication systems and any complex real-life application systems.

2.6 Structure of C++ Program

As it can be seen from the Program 2.3, a typical C++ program would contain four sections as shown in Fig. 2.3. These sections may be placed in separate code files and then compiled independently or jointly.

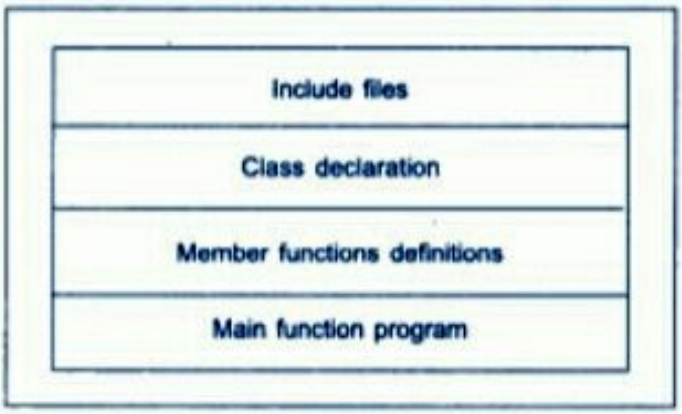

Fig. 2.3 ⇔ *Structure of a C++ program*

It is a common practice to organize a program into three separate files. 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 definition). 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.

This approach is based on the concept of client-server model as shown in Fig. 2.4. The class definition including the member functions constitute the server that provides services to the main program known as client. The client uses the server through the public interface of the class.

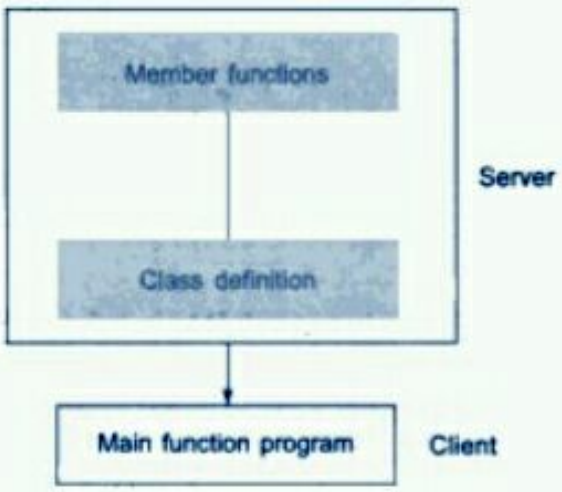

Fig. 2.4 ⇔ *The client-server model*

⊛ Input and Output Operator :

Input operator → For input operator we used scanf() in C but in C++ we use cin>> instead of scanf(). In scanf we used double quotation ("") and address (&) operator before variable in which data is to be stored but we do not use both of them in C++.

For example

```
cin>>number1;
```

In this number1 is variable in which data is stored.

Date.
Page No.

~~Ops~~ Output operator → For output operator in C we used printf() but in C++ we use cout<< and used as follows in C++ program:

cout << "statement" << printing variable;

(#) Character set and Tokens:

(a) Character set → C++ character denotes any alphabet, digit or special symbol, digit or special symbol used to represent information. C uses the upper case letters "A-Z" and lower case letters "a-z", the digits "0-9" and certain special characters (constants, variables, operators, expressions etc.)

(b) Tokens → The smallest individual units in a program are known as tokens. These are the building blocks in C++ language or any other language which are constructed together to write a program. These are of 6 types which are as follows:-

1) Keywords → There are certain words that are reserved for doing ~~special~~ specific tasks, these words are known as keywords and they have standard predefined meaning in C++. These reserved keywords are always written in lower case. There are 48 keywords in C++ and additional 15 keywords by ANSI.
for e.g. int, float, do, while, if, else, switch etc.

Preprocessor Directives: (About #include and define directives)

Write → Link part with some description of define directives from this chapter.

i) **Identifiers** → All the words that we will use in our C++ program will be either keywords or identifiers. Keywords are predefined and can not be changed by the user, while identifiers are user defined words and are used to give names to entities, arrays, functions, structures etc.

ii) **Constants** → Constant is a value that can not be changed during execution of program. These fix values are also called literals. These constants are of three types: numeric, character and string constants. Integer constants include integer constant or floating point constants for eg. x=5, y=2.5 etc. Character constants include lower case letters "a-z" and upper case letters "A-Z". Character constants are always written in single quotation and string in double quotation.

Also called Define directives

Symbolic constants → If we want to use a constant several times then we provide it a name. For e.g. If we had to use the constant 3.14159265 at many places in our program, then we can give it a name pi and use this name instead of writing the constant value. These types of constants are called symbolic constant or named constant.

Syntax: # define name value

for e.g. # define pi 3.14

iv) **Strings** → String is an array of characters or alphabets. Each programming language has a set of chacter to communicate with computers. Strings may be alphabet, digits or special characters written in double quotation.

cin >> variable;
cout << "statement" << variable;

These two are syntax for input (cin) and output (cout).

v) Operators → C++ has a rich set of operators. All C operators are valid in C++. In addition, C++ introduces some new operators, these are:

- << → insertion operator.
- >> → extraction operator
- :: → scope resolution operator.
- delete → Memory release operator.
- endl → line feed operator.
- new → memory allocation operator.
- setw → field width operator.

vi) Special symbols → There are some special characters or symbols in C++ such as (), {}, [], ;, :, /, etc.

(x) Data Types:-

There are mainly three types of data types which are user-defined data-type, basic data types and derived data type.

@ User-defined data type → We have user-defined data types such as struct and union in C. These data types are legal in C++, some more features have been added to make them suitable for object-oriented programming. C++ also permits us to define another user-defined data type known as class which can be used, just like any other basic data type, to declare variables. The class variables are known as objects, which are the central focus of object-oriented programming.

Date.
Page No.

Enumerated data type → An enumerated data type is another user-defined type which provides a way for attaching names to numbers, The enum keyword (from C) automatically enumerates a list of words by assigning them values 0, 1, 2, and so on. The syntax of an enum statement is similar to that of the struct statement.

Examples:

```
enum shape {circle, square, triangle};
enum colour {red, bule, green, yellow};
enum position {off, on};
```

(6) Basic data types / Built-in data type:

Basic data types / built-in data type contain Internal type int and char, void & floating type float and double. ANSI C++ committee has added two more data types bool and wchar_t. The normal uses of void are:

i) To specify the return type of a function when it is not returning any value.

ii) To indicate an empty argument list to a function.

Example: void function1 (void);

Size and range of some of C++ basic data types are as follows:

Type	Bytes	Range
char	1	-128 to 127
int	2	-32768 to 32767
float	4	3.4E-38 to 3.4E+38
double	8	1.7E-308 to 1.7E+308
short int	2	-32768 to 32767
long int	4	-2147483648 to 2147483647

Date.
Page No.

⑥ Derived data types →

i) **Arrays** → The application of arrays in C++ is similar to that in C. The only exception is the way character arrays are initialized. When initilizing a character array in ANSI C, the compiler will allow us to declare the array size as the exact length of the string constant.

For instance,

```
char string[3]="xyz"; //for C
```

is a valid in ANSI C. It assumes that the programmer intends to leave out the null character \0 in the definition. But in C++, the size should be one larger than the number of characters in the string.

```
char string[4]="xyz"; //for C++
```

ii) **Functions** → Functions have undergone major changes in C++. While some of these changes are simple, others require a new way of thinking when organizing our programs. Many of these modifications and improvements were driven by the requirements of the object-oriented concept of C++. Some of these were introduced to make the C++ program more reliable and readable.

iii) **Pointers** → Pointers are declared and initialized as in C. Examples:

```
int *ip; //int pointer
ip=&x; //address of x assigned to ip.
*ip=10; //10 assigned to x through indirection.
```

Date.
Page No.

Imp

⊛ Type Conversion:

Type conversion or typecasting refers to changing an entity of one datatype into another. There are two types of conversion implicit and explicit. It is a process in which variable in one variable in one data type is converted to other datatype.

It is the method of changing an entity from one data type to another. It is used in computer programming to ensure variables are correctly processed by a function. An example of type casting is converting a floating point number to an integer. This might be used to perform calculations more effectively when the decimal precision is unnecessary.

Types:

ⓐ Implicit conversion:-

It is also known as automatic type conversion. It is done by compiler on its own, without any external trigger from the user. Generally it takes place when there is more than one data type in an expression. In such condition type conversion takes place to avoid lose of data. In this all the data types of the variables are upgraded (converted) to the data type of variable with largest data type.

Let us take an example as follows:

```
int m = 15;
float x = 3.1, y;
y = m * x;
```

→ It is type float (4 bytes) higher data type

→ It is type int (2 bytes) lower data type

In the example lower data type int will be automatically converted to float. This type of conversion is implicit conversion.

(b) Explicit conversion:-

This process is also called type casting and it is user-defined conversion of data type. Here the user can typecast (change/convert) the result to make it of a particular data type. In C++, it can be done by two ways:

i) Converting by assignment:- This is done by explicitly (directly) defining the required type infront of the expression in parenthesis. This can be also considered as forcefull casting.

Syntax: (type) expression

where type indicates the data type to which the final result is converted.

For example:

```
int m=5;
float y;
y = (float)m/2;
```

If we do not provide float before variable m as in example above we will not get exact answer as 2.5 instead we get only integer value as 2. So, it is necessary while programming.

ii) Conversion using cast operator:-

A cast operator is a special operator that forces one data type to be converted into another.

Note:- While using manipulator in program for endl we can use directly as "endl" but while using setw() we should include #include<iomanip> preprocessor file.

As an operator, a cast operator is unary operator and has the same precedence as any other unary operator. The most general cast operators supported by C++ compilers are as follows:-

* Static Cast
* Dynamic Cast
* Const Cast
* Reinterpret Cast

Imp

⊛ Manipulators:

Manipulators are operators that are used to format the data display. The most commonly used manipulators are endl and setw.

endl → The endl manipulator works same as the "\n" (i.e; newline) in C. The endl manipulator, when used in an output statement, causes a linefeed to be inserted.

For example:
```
cout<<"m=1"<<endl;
cout<<"n=2"<<endl;
```
This endl in above example will cause to print m=1 in one line and n=2 in next line.

setw → The setw manipulator helps to shift the output statements to right. It is used for shifting output rightwards according to need.

For example:
```
cout<<setw(5)<<sum<<endl;
```
The manipulator setw(5) in above example will right shift (right-justified) the output value of sum in the screen.

Note:- We can also use other manipulators of C in C++ also like "\t".

Date:
Page No.

⑧ Dynamic memory allocation with new and delete:

The process of allocating memory during execution time/runtime of program using heap space of memory to reduce wastage of memory is called dynamic memory allocation.

new operator → The new operator denotes a request for memory allocation on the Heap. If sufficient memory is available, new operator initializes the memory and returns the address of the newly allocated and initialized memory to the pointer variable.

Syntax to use new operator: To allocate memory of any data type, the syntax is:

pointer_variable = new data_type;

Here, pointer_variable is the pointer of type data_type. Data_type could be any built-in data type including array or user defined data types including structure and class.

Example:

```
// Pointer initialized with null
// Then request memory for variable
int *p = null;
p = new int;
```

We can use new operator for initializing memory (for example: int *p = new int(25);) or allocating block of memory (for example: int *p = new int[10]) and for others too.

Date.
Page No.

delete operator →

Since it is programmer's responsibility to deallocate dynamically allocated memory, programmers are provided delete operator by C++ language.

Syntax: delete pointer_variable;

Here, pointer variable is the pointer that points to the data object created by new.

Examples:

```
delete p;
delete q;
```

To free dynamically allocated array pointed by pointer_variable, use following form of delete:

```
delete[] pointer_variable;
```

Example:

```
delete[]p;
```

Control Statements: (Not Imp).

The statements that alter the normal flow of program execution are known as control statements. In some programs we may want to execute only a part of program or we may want to execute some statements several times, this is possible with the help of different control statements. Control statements define how the control is transfered to other parts of the program. As in C, in C++ also mainly two types of control statements: decision making statements and looping statements as follows:

Date.
Page No.

Decision making statements.

i) Simple if statement
ii) If... else statement
iii) Else... if ladder
iv) Nested if statement
v) Switch case statement
vi) Nested switch case.

Looping statements

i) While loop
ii) do... while loop
iii) for loop
iv) Nested loops.

Note:
If needed read fully about these from C copy of 1st sem of chapter Control Statements.

+ Read concepts of functions & Pointer also from C copy.

Note → The repeating things of C are only for concepts that will be useful in C++ but not important from exam point of view like about control statements.

Date.
Page No.

Imp

⊛ Function Overloading:

Two or more functions having same name but different argument(s) are known as overloaded functions and this mechanism used in program is called function overloading.

In C++ programming, two functions can have same name if number or type of arguments passed are different. These functions having different number or type (or both) of parameters are known as overloaded functions.

Let we consider following 4 functions having same name but different arguments or different number of arguments also maybe as follows:

```
int test() { }
int test(int a) { }
float test(double a) { }
int test(int a, double b) { }
```

Here, all 4 functions are overloaded functions because argument(s) passed to these functions are different.

Notice that, the return type of all these 4 functions are not same. Overloaded functions may or may not have different return type but it should have different argument(s).

```
// Error code
int test(int a) { }
double test(int b) { }
```

The number and type of arguments passed to these two functions are same even though the return type is different. Hence, the compiler will throw error.

Date.
Page No.

Example:

```
#include <iostream>
using namespace std;

void display (int);
void display (float);
void display (int, float);

int main() {
        int a=5;
        float b=5.5;

        display (a);
        display (b);
        display (a,b);
        return 0;
      }

void display (int var)
      {
        cout << "Integer number: " << var << endl;
      }
void display (float var)
      {
        cout << " Float number: " << var << endl;
void display (int var1, float var2)
      {
        cout << "Integer number:" << var1;
        cout << " and float number:" << var2;
      }
```

Output

```
Integer number: 5
Float number: 5.5
Integer number: 5 and float number: 5.5
```

Date.
Page No.

Imp

⊛ Inline functions:

The inline functions are a C++ enhancement feature to increase the execution time of a program. Functions can be instructed to compiler to make them inline so that compiler can replace those function definition wherever those are being called. Compiler replaces the definition of inline functions at compile time instead of referring function definition at runtime.

Note:- It is only applicable only for small functions. If function is big then, compiler can ignore the "inline" request and treat the function as normal function.

To make any function as inline, we start its definitions with the keyword "inline".

The syntax for defining function inline is:

```
inline return_type function_name (parameter(s))
{
// Function code
}
```

The following program demonstrates the use of inline function.

```
#include <iostream>
using namespace std;
inline int cube (int s){
return s*s*s;
}
int main(){
cout << "The cube of 3 is" << cube(3);
return 0;
}
```

Output → The cube of 3 is 27.

Date.
Page No.

Advantages:

i) Function call overhead doesn't occur.
ii) It also saves the overhead of push/pop variables on the stack when function is called.
iii) It also saves overhead of a return call from a function.
iv) Inline function may be useful for embedded systems because inline can yeild less code than the function call and return.

Disadvantages:

i) If too many inline functions are used then the size of the binary executable file will be large, because of the duplication of same code.

ii) Too much inlining may reduce the speed of instruction fetch from that of cache memory to that of primary memory.

iii) Inline functions may not be useful for many embedded systems. Because in embedded systems code size is more important than speed.

iv) Inline functions might cause thrashing because inlining might increase size of the binary executable file. Thrashing in memory causes performance of computer to degrade.

Date.
Page No.

⊛ Default Arguments:

A default argument is a value provided in a function declaration that is automatically assigned by the compiler if the caller of the function doesn't provide a value for the argument with a default value.

A default argument is a function argument that has a default value provided to it. If the user does not supply a value for this argument, the default value will be used. If the user supply a value for the default argument, the user-supplied value is used.

Some examples of function declaration with default values are:

```
int add (int x, int y=20, int z=30);   //Valid
int add (int x=10, int y=20, int z=30); // Valid

int add (int x=10, int y, int z);  //Invalid
int add (int x=10, int y, int z=30); //Invalid
```

The last two examples of function declaration are invalid since only the trailing arguments can have default values and therefore we must add default values from right to left. We can not provide a default values to a particular argument in the middle of an argument list.

Date.
Page No.

Example of default argument value.

```
#include<iostream.h>
#include<conio.h>

int add(int x, int y=20, int z=30)
    {
      return x+y+z;
    }

void main()
    {  int rs;
       rs = add(5);
      cout<<"The sum is:"<<rs;endl;

       rs=add(4,5);
       cout<<"The sum is:"<<rs;endl;

       rs=add(7,3,4);
      cout<<"The sum is:"<<rs;endl;
    }
```

Output:

```
The sum is: 55
The sum is: 42
The sum is: 14
```

Date
Page No.

Imp

8. Call by Refrence / Pass by Refrence:

There may arise a situation where we would like to change the values of variables in the calling program using function which is not possible if the call-by-value method is used. This is because the function does not have access to the actual variables in the calling program and can only work on the copy of the values. Call-by-value method is fine if the function does not need to alter the values of the original variables in calling program.

Example for call-by-value

```
#include<iostream>
using namespace std;

void change(int);
int main(){
        int a=15;
        clrscr();
cout<<"Before calling function a=";<<a<<endl;
        change(a);
cout<<"After calling function a=";<<a<<endl;
        getch();
}
void change(int x)
{
        return x+5;
}
```

Output:

Before calling function a=15
After calling function a=15

Note:- In additional read about Scope/~~Urbi~~ Visibility of variables and Concept of Pointers in arrays and functions for C copy of 1st sem.

In this example only the copy of actual parameter into formal parameter is made. So, changes made on the value is not reflected to the original value of calling function (main function). So we need to use call-by-refrence if we want to access original value and want to make changes on those values.

In call by refrence we pass the address or location of a variable to the function. Pointers are used to call a function by refrence. When a function is called by refrence, the the formal arguments becomes refrences to the actual arguments.

Example for call by refrence

```
#include<iostream>
using namespace std;
void change(int*);

int main(){                          (actual argument)
    int a=15;
    clrscr();
cout<<"Before calling function a="<<a;endl;
    change(&a);          (address sent)
cout<<"After calling function a="<<a;endl;
    getch();
}
void change(int*x){          (formal argument)
    *x=*x+5;
}
```

Output:

Before calling function a=15
After calling function a=20

Date.
Page No.

Imp

(5) Scope Resolution Operator (::):

Blocks and scopes can be used in constructing programs. We know that the same variable name can be used to have different meanings in different blocks. The scope of the variables extends from the point of its declaration till the end of the block containing the declaration. A variable declared inside a block is said to be local to that block.

In C the global version of a variable cannot be accessed from inner block. C++ resolves this problem by introducing new operator :: called scope resolution operator. This can be used to uncover a hidden variable. It takes following form:

```
:: variable_name
```

The scope resolution operator (::) basically does following two things:

1) Access the global variable when we have a local variable with same name.

Example:

```
#include <iostream>
using namespace std;
int a=100;  //global variable

int main(){
    int a=200;  //local variable
cout<<"local variable:"<<a<<endl;
cout<<"Global variable:"<<::a<<endl; //using scope resolution operator

    return 0;
}
```

Output:

Local variable: 200
Global variable: 100

If we did not used :: operator then Output will be 200 for both the values of a.

Date.
Page No.

2) Define a function outside the class:

Let we declare two functions user1() and user2() inside the class named check. Let us define the functions outside the class.

Since user1() and user2() are the member functions of the class check, we need to use a scope resolution operator to define the function outside the check class, as follows:

```
#include<iostream>
using namespace std;

class check {
    public:
    void user1(); //function declaration
    void user2();
};
// function defination outside the class using :: operator.

void check::user1(){
    cout<<"I am user 1"<<endl;
}

void check::user2(){
    cout<<"I am user 2"<<endl;
}

int main(){
    check user;
    user.user1();
    user.user2();
    return 0;
}
```

Output:

```
I am user 1
I am user 2.
```

not important with exam point of view.

Date.
Page No.

Type Compatibility:

C++ is very strict with regard to type compatibility as compared to C. For instance, C++ defines int, short int and long int as three different types. They must be cast when their values are assigned to one another.

Similarly, unsigned char, char and signed char are considered as different types, although each of these has a size of one byte. In C++, the types of values must be the same for complete compatibility or else, a cast must be applied. These restrictions in C++ are necessary in order to support function overloading where two functions with the same name are distinguished using the type of function arguements.

The type compability is categorized into following three types by the compiler.

1) Assignment compatibility

For example:
```
float n1=12.5;
int n2=n1;
```
This assigning of float value to int type will result in loss of decimal value of n1. However this type compatibility will not show any error but might give a warning "possible loss of data".

2) Expression compatibility

Example:
```
int n=3/2;
cout<<n;
```
Here the result will be 1. The actual result of 3/2 is 1.5 but because of incompatibility there will be loss of decimal value.

Date.
Page No.

9) Parameter compatibility

Due to incompatibility in type of actual parameter and formal parameters loss of data occurs.

For example:

```
void show (int n)
{
    cout << "n=" << n;
}
void main( )
{
    show (8.2);
}
```

Here output will be n=8 due to incompatibility in actual and formal parameter type.