MARUDHAR KESARI JAIN COLLEGE FOR WOMEN,VANIYAMBADI

PG AND RESEARCH DEPARTMENT OF COMPUTER SCIENCE

CLASS: I BSC COMPUTER SCIENCE

SUBJECT CODE: 23UCS11

SUBJECT NAME: Object Oriented Programming Concepts using C++

SYLLABUS

UNIT I

Introduction to C++ - key concepts of Object-Oriented Programming – Advantages – Object Oriented Languages – I/O in C++ - C++ Declarations. Control Structures : - Decision Making and Statements : If ..else, jump, goto, break, continue, Switch case statements - Loops in C++ :for, while, do - functions in C++ - inline functions – Function Overloading

Introduction of c++

C++ is a general-purpose programming language that was developed as an enhancement of the C language to include object-oriented paradigm. It is an imperative and a compiled language.

1. C++ is a high-level, general-purpose programming language designed for system and application programming. It was developed by Bjarne Stroustrup at Bell Labs in 1983 as an extension of the C programming language. C++ is an object-oriented, multi-paradigm language that supports procedural, functional, and generic programming styles.
2. One of the key features of C++ is its ability to support low-level, system-level programming, making it suitable for developing operating systems, device drivers, and other system software. At the same time, C++ also provides a rich set of libraries and features for high-level application programming, making it a popular choice for developing desktop applications, video games, and other complex applications.
3. C++ has a large, active community of developers and users, and a wealth of resources and tools available for learning and using the language. Some of the key features of C++ include:
4. Object-Oriented Programming: C++ supports object-oriented programming, allowing developers to create classes and objects and to define methods and properties for these objects.
5. Templates: C++ templates allow developers to write generic code that can work with any data type, making it easier to write reusable and flexible code.
6. Standard Template Library (STL): The STL provides a wide range of containers and algorithms for working with data, making it easier to write efficient and effective code.
7. Exception Handling: C++ provides robust exception handling capabilities, making it easier to write code that can handle errors and unexpected situations.

Key concepts of object oriented programming

- **Objects**
- **Classes**
- **Data abstraction and encapsulation**
- **Inheritance**
- **Polymorphism**
- **Dynamic binding**
- **Message passing**

Objects

Objects are the basic run time entities in an object-oriented system. They may represent a person, a place, a bank account, a table of data or any item that the program has to handle. They may also represent user-defined data such as vectors, time and lists. Objects take up space in the memory and have an associated address like a record in Pascal, or a structure in c.

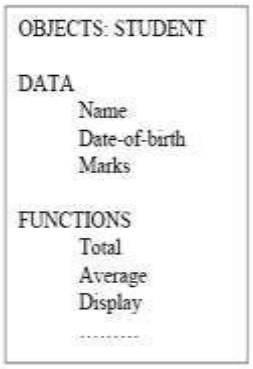


Fig. 1.5 representing an object

Classes

The entire set of data and code of an object can be made a user-defined data type with the help of class. In fact, objects are variables of the type class. Once a class has been defined, we can create any number of objects belonging to that class. Each object is associated with the data of type class with which they are created. A class is thus a collection of objects similar types.

For examples, Mango, Apple and orange members of class fruit.

Classes are user-defined that types and behave like the built-in types of a programming language. If fruit has been defines as a class, then the statement

Fruit Mango;

Will create an object **mango** belonging to the class **fruit.**

Data Abstraction and Encapsulation

The wrapping up of data and function into a single unit (called class) is known as ***encapsulation.*** The data is not accessible to the outside world, and only those functions which are wrapped in the class can access it. These functions provide the interface between the object‟s data and the program. This insulation of the data from direct access by the program is called ***data hiding or information hiding***.

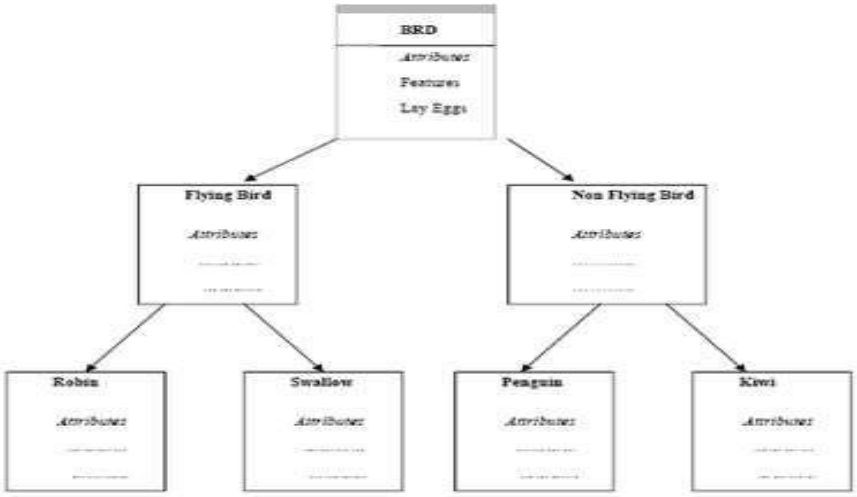
Abstraction refers to the act of representing essential features without including the background details or explanation. They encapsulate all the essential properties of the object that are to be created. The attributes are some time called **data members** because they hold information. The functions that operate on these data are sometimes called **methods or member function.**

Inheritance

Inheritance is the process by which objects of one class acquired the properties of objects of another classes. It supports the concept of hierarchical classification. For example, the bird,„robin‟ is a part of class „flying bird‟ which is again a part of the class „bird‟. In OOP, the concept of inheritance provides the idea of reusability.

This means that we can add additional features to an existing class without modifying it.

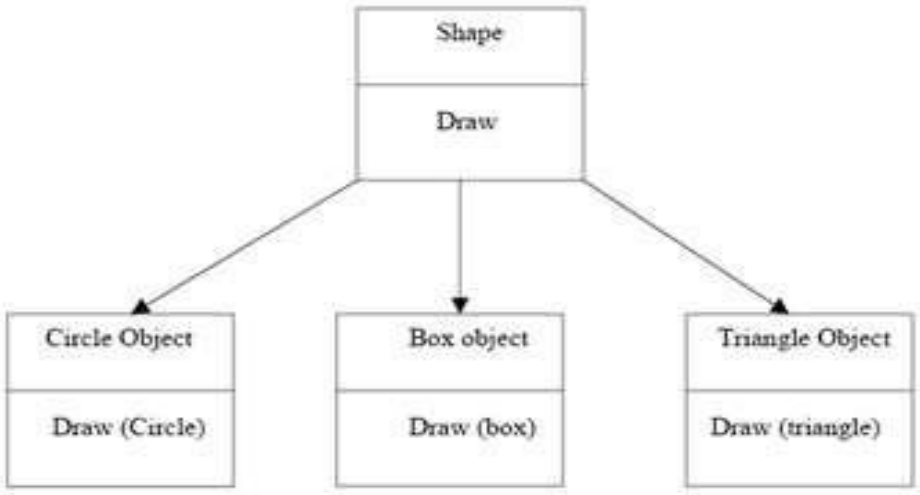
Property inheritances



Polymorphism

Polymorphism is another important OOP concept. Polymorphism, a Greek term, means the ability to take more than on form. An operation may exhibit different behavior is different instances.

Polymorphism is extensively used in implementing inheritance.



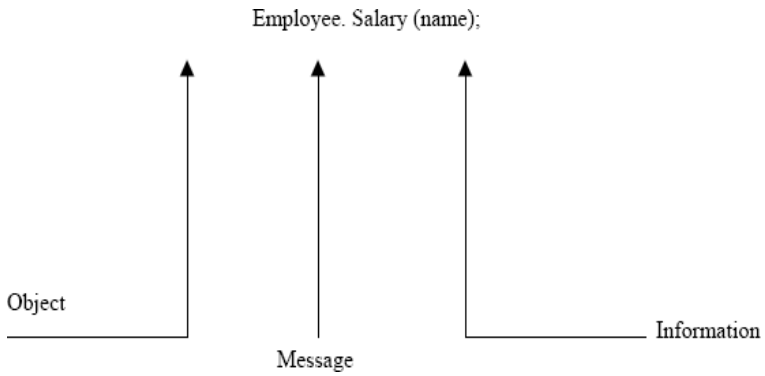
Dynamic Binding

Binding refers to the linking of a procedure call to the code to be executed in response to the call. Dynamic binding means that the code associated with a given procedure call is not known until the time of the call at run time. It is associated with polymorphism and inheritance.

Message Passing

An object-oriented program consists of a set of objects that communicate with each other. The process of programming in an object-oriented language, involves the following basic steps:

1. Creating classes that define object and their behavior,
2. Creating objects from class definitions, and
3. Establishing communication among objects.



Objects communicate with one another by sending and receiving information much the same way as people pass messages to one another. M*essage passing* involves specifying the name of object, the name of the function (message) and the information to be sent. Object has a life cycle. They can be created and destroyed.

Advantages of OOPs:

The principal advantages are:

- Through inheritance, we can eliminate redundant code extend the use of existing Classes.
- We can build programs from the standard working modules that communicate with one another, rather than having to start writing the code from scratch. This leads to saving of development time and higher productivity.
- The principle of data hiding helps the programmer to build secure program that cannot be invaded by code in other parts of a programs.
- It is possible to have multiple instances of an object to co-exist without any interference.
- It is possible to map object in the problem domain to those in the program.
- It is easy to partition the work in a project based on objects.
- The data-centered design approach enables us to capture more detail of a model can implemental form.
- Object-oriented system can be easily upgraded from small to large system.

Object Oriented Language

The languages should support several of the OOP concepts to claim that they are object-oriented. Depending upon the features they support, they can be classified into the following two categories:

1. Object-based programming languages, and
2. Object-oriented programming languages.

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

- Data encapsulation
- Data hiding and access mechanisms
- Automatic initialization and clear-up of objects
- Operator overloading

Languages that support programming with objects are said to the objects-based programming languages. They do not support inheritance and dynamic binding. Ada is a typical object-based programming language.

Object-oriented programming language incorporates all of object-based programming features along with two additional features, namely, inheritance and dynamic binding. Object-oriented programming can therefore be characterized by the following statements:
Object-based features + inheritance + dynamic binding

INPUT OUTPUT IN C++:

- **Input Stream:** If the direction of flow of bytes is from the device(for example, Keyboard) to the main memory then this process is called input.
- **Output Stream:** If the direction of flow of bytes is opposite, i.e. from main memory to device(display screen) then this process is called output.

Header files available in C++ for Input/Output operations are:

1. **iostream**: iostream stands for standard input-output stream. This header file contains definitions to objects like cin, cout, cerr etc.
2. **iomanip**: iomanip stands for input output manipulators. The methods declared in this files are used for manipulating streams. This file contains definitions of setw, setprecision etc.

3. **fstream**: This header file mainly describes the file stream. This header file is used to handle the data being read from a file as input or data being written into the file as output.

Standard output stream (cout): Usually the standard output device is the display screen. The C++ **cout** statement is the instance of the ostream class. It is used to produce output on the standard output device which is usually the display screen. The data needed to be displayed on the screen is inserted in the standard output stream (cout) using the insertion operator(<<).*filter_none*

```
#include <iostream>

using namespace std;


int main()

{

   char sample[] = "GeeksforGeeks";


   cout << sample << " - A computer science portal for          geeks" << endl;


   return 0;

}
```

standard input stream (cin): Usually the input device in a computer is the keyboard. C++ cin statement is the instance of the class **istream** and is used to read input from the standard input device which is usually a keyboard.

The extraction operator(>>) is used along with the object **cin** for reading inputs. The extraction operator extracts the data from the object **cin** which is entered using the keboard.

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

int main()
{
   int age;

   cout << "Enter your age:";
   cin >> age;
   cout << "Your age is: " << age << endl;

   return 0;
}
```

buffered standard error stream (clog): This is also an instance of io stream class and used to display errors but unlike cerr the error is first inserted into a buffer and is stored in the buffer until it is not fully filled. The error message will be displayed on the screen too.*filter_none*

```
1. #include <iostream>
2. using namespace std;
3.
4. int main()
5. {
6.     clog << "An error occured";
7.     return 0;
8. }
```

C++ Tokens

In C++ every statement is made up of smallest individual elements called tokens. Tokens are the basic lexical elements. Tokens in C++ include the following

- Keywords
- Identifiers
- Constants
- Operator *Data types

Keywords

The list given below contains all the C++ keywords. Every keyword will perform a specific operation in C++. Keywords in C++ cannot be redefined by a programmer; further keywords cannot be used as identifier name.

Identifiers

Every named program component in C++ like variables, functions, arrays, classes etc. must have legal identifier (name) .In C++ all variables must be declared before they are used. Furthermore, variables must be used in a manner consistent with their associated type. An identifier is formed with a sequence of letters (including „_ „) and digits. The first character must be a letter. Identifiers are case sensitive, i.e., first_name is different from First_name.

Examples of valid variable names are

myname, jj, name123

Examples of invalid variables

- 12B Invalid variable name since its starts with a number.
- Basic pay Invalid variable name since it has a space
- ABC,1 Invalid variable name since , is a special character

Constants

Programming in c++

15

Constants refer to values that do not change during the execution of the program.

Examples for constants are

1998 Integer constant 25.789 Floating point constant

“Hello World” String constant „a“ Character constant

Data Types

C++ supports three categories of data types. They are

- Built in Data types
- User Defined Data types
- Derived Data types

Size and Range of basic data types

Built in Data Types

C++ supports the following built in data types. They are

Signed integer:

The signed integer types are short int, int and long int. Optionally the keyword signed can be used in front of these definitions, e.g. signed int. The size of these variables depends on the implementation of the compiler. The only rule for the length of short, int, long is: short<= int<=long, i. e. they may have all the same size. Examples:

□□int x,y

□□long int a;

Unsigned integer:

The unsigned integer types are unsigned short int (abbreviation unsigned short), unsigned int and unsigned long int (abbreviation unsigned long). The sizes follow the same rules as for singed integers.

Examples

□□unsigned short a;

□□unsigned long x;

Character type:

Character type variables can hold a single character. The character types are char, signed char and unsigned char. Note that a simple char can mean either signed or unsigned, as

this is not defined in ANSI C. It depends therefore only on the implementation of the compiler. A variable can be declared as character type as given below. char x

Floating point types:

The floating point types are float, double and long double. A float variable has a single precision value. A double or long double variable has at least single precision, but often has double precision (however, this depends on the implementation). The rule for the size of floating point types is: float <= double<=long double. Examples for floating point variable declarations are listed below.

□□double m

□□float f

Boolean type:

Standard C++ has an explicit boolean type bool. (C only has an implicit boolean type: 0 = logical false, nonzero = logical true). This type is used to express results of logical operations. The value of the Boolean type is symbolic, either false or true. An integer value of 0 indicates false and a non-zero value indicates true. A variable can be declared as boolean type as given below.

□□bool logic

Empty/void type:

The special type void is used for three occasions: defining a function without parameters, defining a function that has no return value and for generic pointers. Example

void func()

User Defined Data Types

Structures, Unions and Classes

A structure is a collection of one or more variables; the variables may belong to different types, grouped together under a single name for easy handling. A structure in C++ can be declared as

```
struct point
{
int x;
int y; };
```

The keyword struct introduces a structure declaration, which is a list of declarations enclosed in braces.

A struct declaration defines a user defined type. The close brace that terminates the list of members may be followed by a list of variables, just as for any basic type.

Union

A union is a user defined data type that may hold (at different times) objects of different types and sizes, with the compiler keeping track of size and alignment requirements. Unions provide a way to manipulate different kinds of data in single area of storage, without

embedding any machine-dependent information in the program. The syntax is based on structures:

```
union u1
{
int i;
float f;
char c;
} u;
```

The variable u will be large enough to hold the largest of the three types; the specific size is implementation-dependent.

Unions allow only one variable to refer at a time; all union variables cannot be accessed simultaneously. Syntactically, members of a union are accessed as ***union-name. member*** or ***union-pointer->member.***

Enumeration Data Type

An enumeration is a set of named integer constants that specify all the allowable set of values a variable of that type may have.The general syntax of enumeration data type is

enum enum-name{enumeration values} ;

Example

enum states{Andhara,Karnataka,Kerala, Tamilnadu} ;

enum currency { dollar,euro,rupees}

Derived Data Type

Arrays

In C++ arrays are handled in the same way as of C. For example the declaration

int a[10]; defines an array of size 10, that is, a block of 10 consecutive objects named a[0], a[1], ...,a[9].

The notation a[i] refers to the i-th element of the array.

Pointers

A pointer is a variable that contains the address of a variable. Pointers in C++ are declared and initialized as in C.

Examples

int a= 1, b= 2

int *ip; /* ip is a pointer to int */

ip = &a /* ip now points to a*/

b= *ip; /* bis now 1 */

ip = 0; / a is now 0 */

Symbolic Constants

There are two ways of creating symbolic constants in C++:

- Using the qualifier **const**, and
- Defining a set of integer constants using **enum** keyword.

A const-qualified object allows you to specify the exact type of the constant. For example,

const unsigned int buffer_size = 256;

const int size=10;

Another method of naming integer constant is by enumeration as under;

Enum{X,Y,Z}; This defines X, Y and Z as integer constants with values 0,1 and 2 respectively. This is equivalent to

Const X=0;

Const Y=1;

Const Z=2;

Reference Variables

A reference variable is an alias, that is, another name for an already existing variable. Once a reference is initialized with a variable, either the variable name or the reference name may be used to refer to the variable.

Syntax: ***data-type & reference-name=variable name;***

Example:

Float total=100;

Float &sum=total;

Dynamic initialization

Dynamic initialization means the first value assigned to the variable after memory allocation is not known at compile time, it is evaluated only at run time. For example

const int t = sample(); //dynamic initialization

I nt p = sample(); //dynamic initialization

Scope resolution operator in c++

Scope resolution operator(::) is used to define a function outside a class or when we want to use a global variable but also has a local variable with same name.

Example: 1

….

….

{

Programming in c++

19

int x = 10;

….

….

{

int x = 20;

…..

….

}

….

}

The declaration of the inner block hides the declaration of same variable in outer block. This means, within the inner block, the variable x will refer to the data object declared therein. To access the global version of the variable, C++ provides scope resolution operator. In the above example, x has a value of 20 but ::x has value 10.

Example: 2

```
#include <iostream>
using namespace std;
char c = 'a'; // global variable
int main() {
char c = 'b'; //local variable
cout << "Local c: " << c << "\n";
cout << "Global c: " << ::c << "\n"; //using scope resolution operator
return 0;
}
```

Member dereferencing operators

C++ lets you define pointers to members of a class. These pointers involve special notations to declare them and to dereference them.

Operator Function

::* To declare a pointer to a member of a class

* To access a member using object name and a pointer to that member

>* To access a member using a pointer to the object and a pointer to that member

Memory Management Operators

There are two types of memory management operators in C++:

□□new

□□delete

These two memory management operators are used for allocating and freeing memory blocks in efficient and convenient ways.

Control Structures

The control-flow of a language specifies the order in which computations are performed. The various control structures supported by C++ is discussed below.

If-Else

The if-else statement is used for making decisions. The syntax is

if (expression)

statement1

else

statement2

The flow chart for if else is given below.

Where the else part is optional. The expression is evaluated; if it is true (that is, if *expression* has a non-zero value), statement1 is executed. If it is false *(*expression *is zero)* and if there is an else part, *statement2* is executed instead.

Example

```
if(a>b)
c=a;
else
c=b;
```

In this example the value of a will be assigned c if a is greater than b else the value of b will be assigned to c.

Else if

The syntax for else-if ladder structure is given below

if (expression)

statement

else if (expression)

statement

else if (expression)

statement

else if (expression)

statement

else

statement

This sequence of if statements is the most general way of writing a multi-way decision. The expressions are evaluated in order; if an expression is true, the statement associated with it is executed, and this terminates the whole chain. The last else part handles the default case where none of the other conditions is satisfied. An example for this construct is given below.

```
if (( a>b) && (a>c))
cout<<"A is big";
else if ((b>a) && (b>c))
cout<<"B is big";
else if ((c>a) && (c>b))
```

cout<<"C is big";

Switch Statement

The switch statement is a multi-way decision that tests whether an expression matches one of a number of *constant* integer values, and branches accordingly. The syntax of switch statement is given below.

switch (*expression*)

{

case *const-expr*:

statements

case *const-expr*:

statements

default:

statements

}

Each case is labeled by one or more integer-valued constants or constant expressions.

If a case matches the expression value, execution starts at that case. A default is optional; it gets executed if none of the cases is matched. Cases and the default clause can occur in any order. An example for switch case construct is given below.

```
switch(k)
{
case 1:
cout<< "The value of k is 1" ;
break;
case 2:
cout<< "The value of k is 1" ;
break;
case 3:
cout<< "The value of k is 1" ;
break;
default:
cout<<" The value of k is other than 1,2,3" ;
}
```

Here based on the value of k case 1, case 2 or case 3 will be executed .If the value of k is other than 1, 2 and 3 then default case will be executed. The break statement causes an immediate exit from the switch after the code for one case is done, execution *falls through* to the next case unless you leave the switch .You can use break statement to explicitly come out of the switch.Programming in c++26

Looping Statements

C++ supports three type of looping statements as C language .They are

- While Loop
- Do…While Loop
- For Loop

While Loop

The Syntax of while loop is

while (expression)

{

statements ;

}

the *expression* is evaluated. If it is non-zero(i.e. if the condition given in the expression is true) , statement*s* is executed and expression is re-evaluated. This cycle continues until *expression* becomes zero (false).

Example

```
i=0; //initial value
while(i<100) //condition
{
cout<<i;
i=i+2 //increment
}
```

Here in this example the while loop prints all the even numbers between 0 and 100.Note the while loop checks for the condition i<100 every time when the loop iterates.

For Loop

The for statement is given below

for (expr1; expr2; expr3)

{

statements ;

}

is equivalent to the while statement given in the previous example

expr1

while (expr2)

{

statement

expr3;

}

Most commonly, *expr1* and *expr3* are assignments or function calls and *expr2* is a relational expression. Any of the three parts can be omitted, although the semicolons must remain. If *expr1* or *expr3* is omitted, it is simply dropped from the expansion. If the test, *expr2*,

is not present, it is taken as permanently true.

```
for (i=0 ;i<100 ;i=i+2)
cout<<i ;
```

This is the for loop version of the previous program for printing even numbers between 0 and 100.Here we can note that in for loop the initilaization, condition and increment statements are kept together in the for statement.

Do...while Loop

The while and for loops test the termination condition at the top. But the do while, tests at the bottom *after* making each pass through the loop body; the body is always executed at least once. The Syntax of do while statement is

do
{
Statements ;
}while (expression);

The *statement* is executed, and then *expression* is evaluated. If it is true, *statement* is evaluated again, and so on. When the expression becomes false, the loop terminates. do...while loop is essential when we expect the loop to be executed at least once irrespective of the condition.

Example

```
i=0; //initialization
do
{
cout<<i;
i+=2; //increment
}while(i<100); //condition.
```

Functions in c++:

A function is a block of code which only runs when it is called.You can pass data, known as parameters, into a function.Functions are used to perform certain actions, and they are important for reusing code: Define the code once, and use it many times.

Create a Function

C++ provides some pre-defined functions, such as main(), which is used to execute code. But you can also create your own functions to perform certain actions.

To create (often referred to as *declare*) a function, specify the name of the function, followed by parentheses **()**

Inline Functions in C++

C++ provides inline functions to reduce the function call overhead. An inline function is a function that is expanded in line when it is called. When the inline function is called whole code of the inline function gets inserted or substituted at the point of the inline function call. This substitution is performed by the C++ compiler at compile time. An inline function may increase efficiency if it is small.

Syntax:

```
inline return-type function-name(parameters)
{
    // function code
}
```

Inline functions Advantages:

1. Function call overhead doesn't occur.
2. It also saves the overhead of push/pop variables on the stack when a function is called.
3. It also saves the overhead of a return call from a function.
4. When you inline a function, you may enable the compiler to perform context-specific optimization on the body of the function. Such optimizations are not possible for normal function calls. Other optimizations can be obtained by considering the flows of the calling context and the called context.
5. An inline function may be useful (if it is small) for embedded systems because inline can yield less code than the function called preamble and return.

Inline function Disadvantages:

1. The added variables from the inlined function consume additional registers, After the in-lining function if the variable number which is going to use the register increases then they may create overhead on register variable resource utilization.
2. If you use too many inline functions then the size of the binary executable file will be large, because of the duplication of the same code.
3. Too much inlining can also reduce your instruction cache hit rate, thus reducing the speed of instruction fetch from that of cache memory to that of primary memory.
4. The inline function may increase compile time overhead if someone changes the code inside the inline function then all the calling location has to be recompiled because the compiler would be required to replace all the code once again to reflect the changes, otherwise it will continue with old functionality.
5. Inline functions may not be useful for many embedded systems. Because in embedded systems code size is more important than speed.

Function Overloading in C++

Function overloading is a feature of object-oriented programming where two or more functions can have the same name but different parameters.

When a function name is overloaded with different jobs it is called Function Overloading. In Function Overloading "Function" name should be the same and the arguments should be different.

Function overloading can be considered as an example of a polymorphism feature in C++.

- Parameters should have a different type

add(inta,int)

add(double a, double b)

Example Program:

```
#include <iostream>
using namespace std;
void print(int i) {
 cout << " Here is int " << i << endl;
}void print(double  f) {
  cout << " Here is float " << f << endl;
}
void print(char const *c) {
  cout << " Here is char* " << c << endl;
}
int main() {
  print(10);
  print(10.10);
  print("ten");
  return 0;
}
```

Output

```
 Here is int 10
 Here is float 10.1
 Herechar*isten
```