

Chapter 4 :

Function in C++

Reference : E. Balaguruswamy Object Oriented Programming With C++; chapter-4.

Topics:

- **Topic 1: Basic of function**
- **Topic 2: Inline function**
- **Topic 3: Function overloading**
- **Topic 4: Basic C++ program using function**

Questions:

Topic 1: Basic of function

1. What is normal function? What is volatile function?
2. Describe different styles of prototyping. Write the advantages of function prototyping in C++.
3. What do you mean by default argument? When do we need to use default arguments in a function. Explain with an example.
4. Describe the ambiguity in case of default argument with an example.
5. Write down the difference between “call by value” and “call by reference”.

Topic 2: Inline function

6. What is inline function? When will you make a function inline? Situation where inline function does not work?
7. Write the advantages and disadvantages of inline function.
8. How does inline function differ from preprocessor macro?

Topic 3: Function overloading

9. What do you mean by function overloading? When do we need function overloading-explain with an example.

Or, write a program in C++ that finds the area of different shapes using function overloading.

10. Differentiate between function overloading and function overriding.

Topic 4: Basic C++ program using function

11. Write a c++ program that inputs a string from the keyboard and determine the length of that string.

12. Write a c++ program that evaluate the following function to 0.0001% accuracy :

$$sum = 1 + (1/2)^2 + (1/3)^2 + (1/4)^2 + \dots$$

13. Write a c++ program that evaluate the following function to 0.0001% accuracy :

$$sum = 1 + (1/2)^2 + (1/3)^3 + (1/4)^4 + \dots + (1/n)^n$$

14. Write a c++ program that evaluate the following function to 0.0001% accuracy :

$$\sin(x) = x - \frac{x^3}{3!} + \frac{x^5}{5!} - \frac{x^7}{7!} + \dots$$

15. Write a c++ program that evaluate the following function to 0.0001% accuracy :

$$\cos(x) = 1 - \frac{x^2}{2!} + \frac{x^4}{4!} - \frac{x^6}{6!} + \dots$$

Q:1:: What is normal function? What is volatile function?

Answer:

Normal Function : Normal function is basically simply a common function in C++. It promotes code reuse and makes the program modular. During function calls, a lot of overhead tasks are performed like saving registers, pushing arguments to the stack, and returning to the calling function.

Volatile Function :

Volatile is not a function rather it is a quantifier which used with different types variable like- normal variable,const type identifier,object,structure,union,pointer etc.The volatile quantifier applied to a variable when it is declared as : `volatile int count;`

In general, variable value is stored in actual memory and when programmer use the variable then a copy of that variable value comes to cache memory and the compiler uses the variable from cache memory not from the actual memory. When volatile keyword use with the variable then the compiler use the variable value directly from the actual memory not from the cache memory.

Q:2:: Describe different styles of prototyping.Write the advantages of function prototyping in C++.

Answer:

Function Prototyping :

Function prototype is a declaration statement in the calling program and is of the following form: `function_return_type function_name(argument- list);`

Different style of declaring function prototype:

- The argument list contains the types and names of arguments that must be passed to the function.Example-

`float volume (int x, float y , float z);`

Note that each argument variable must be declared independently inside the parentheses.That is a combined declaration like:

`float volume(int x, float y, z);` is illegal.

- In a function declaration the names of the arguments are dummy variables and therefore, they are optional- that is the form:

```
float volume(int, float, float);
```

is acceptable at the place of declaration. At this stage, the compiler only checks for the type of arguments when the function is called.

In general, we can either include or exclude the variable names in the argument list of prototypes. The variable names in the prototype just act as placeholders and therefore, if names are used, they don't have to match the names used in the function call or function definition.

Advantage of function prototyping :

The prototype describes the function interface to the compiler by giving details such as the number and type of arguments and the type of return values. With function prototyping a template is always used when declaring and defining a function. When a function is called, the compiler uses the template to ensure that proper arguments are passed, and the return value is treated correctly. Any violation in matching the arguments or the return types will be caught by the compiler at the time of compilation itself.

Q:3::What do you mean by default argument? When do we need to use default arguments in a function.Explain with an example.

Answer:

C++ allows us to call a function without specifying all its arguments. In such cases, the function assigns a default value to the parameter which does not have a matching argument in the function call. Default values are specified when the function is declared i.e. function prototype as well as function definition. The compiler looks at the prototype to see how many arguments a function uses and alerts in the program for possible default values.

Here is an example:

```
float amount( float principal, int period, float rate = 0.15 );
```

The default value is specified in a manner syntactically similar to a variable initialization. The above prototype declares a default value of 0.15 to the argument rate. A subsequent function call like:

```
value = amount (5000 , 7) ; // one argument missing
```

passes the value of 5000 to principal and 7 to period and then lets the function use default value of 0.15 for rate. But the function call is as:

```
value = amount (5000, 7 , 0.12); // no missing argument
```

passes an explicit value of 0.12 to rate.

The situation when do we need to use default arguments in a function:

Default arguments are useful in situations where some arguments always have the same value. For instance, bank interest may remain the same for all customers for a particular period of deposit. It also provides a greater flexibility to the programmer. A function can be written with more parameters than are required for its most common application. Using default arguments, a programmer can use only those arguments that are meaningful to a particular situation.

Advantage of providing default arguments are:

1. We can use default arguments to add new parameters to the existing functions.
2. Default arguments can be used to combine similar functions into one.

Here is an example program :

Github code-

https://github.com/SyedaJannatul/DIIT_Object-Oriented-Programming/blob/9718bb46e4959dcfe6ad639c88685e659c8895ed/chapter%20functions%20in%20C%2B%2B/chapter%20class/question_3_code.cpp

```

#include<iostream>
#include<math.h>
using namespace std;

float value(float p, int n, float r = 0.15)
{
    float total, rate, temp;
    rate = 1+r;
    temp = pow(rate, n);
    total = p*temp;

    return total;
}

int main()
{
    float amount1, amount2;
    amount1=value(5000, 5);
    amount2=value(10000, 5, .30);

    cout<<"Amount_1 = "<<amount1<<endl;
    cout<<"Amount_2 = "<<amount2<<endl;

    return 0;
}

```

Output of the program:

```

Amount_1 = 10056.8
Amount_2 = 37129.3

Process returned 0 (0x0)   execution time : 0.062 s
Press any key to continue.

```

Q:4::Describe the ambiguity in case of default argument with an example.

Answer:

A default argument is checked for type at the time of declaration and evaluated at the time of call . One important point to note is that only the trailing arguments can have default values and therefore we must add defaults from right to left. We cannot provide a default value to a particular argument in the middle of an argument list.

Some examples of function declaration with default values are:

```
int mul(int i, int j = 5, int k = 0)    //legal
int mul(int i = 4, int j)                //illegal
int mul(int i = 4, int j, int k = 0)     //illegal
int mul(int i = 4, int j = 5, int k = 0) //legal
```

These illegal actions are known as ambiguity of the default arguments function.

Q:5:: Write down the difference between “call by value” and “call by reference”.

Answer:

Parameters	Call by value	Call by reference
Definition	While calling a function, when you pass values by copying variables, it is known as “Call By Values.”	While calling a function, in programming language instead of copying the values of variables, the address of the variables is used it is known as “Call By References.
Arguments	In this method, a copy of the variable is passed.	In this method, a variable itself is passed.
Effect	Changes made in a copy of variable never modify the value of variable outside the function.	Change in the variable also affects the value of the variable outside the function.
Alteration of value	Does not allow you to make any changes in the actual variables.	Allows you to make changes in the values of variables by using function calls.
Passing of variable	Values of variables are passed using a straightforward method.	Pointer variables are required to store the address of variables.
Value modification	Original value not modified.	The original value is modified.
Memory Location	Actual and formal arguments will be created in different memory location	Actual and formal arguments will be created in the same memory location
Safety	Actual arguments remain safe as they cannot be modified accidentally.	Actual arguments are not Safe. They can be accidentally modified, so you need to handle arguments operations carefully.

Call by reference can be perform in two way:

- i) use reference variables as arguments in the function call, then the pointer variables need to be used as parameters in the function definition .
- ii) use reference variables as parameters in the function definition.

Here is an example code to clear the concept of “call by reference” :

Github code -

https://github.com/SyedaJannatul/DIIT_Object-Oriented-Programming/blob/9718bb46e4959dcfe6ad639c88685e659c8895ed/chapter%20_functions%20in%20C%2B%2B/chapter%20_class/question_5_code.cpp


```

#include<iostream>
using namespace std;

void swap1(int *a, int *b)
{
    int temp;
    temp = *a;
    *a = *b;
    *b = temp;
}

void swap2(int &a, int &b) // a and b are reference variable
{
    int temp;
    temp = a;
    a = b;
    b = temp;
}

int main()
{
    int m,n,r,s;
    cout<<"Enter two integer : ";
    cin>>m>>n;
    cout<<"Before swapping m = "<<m<<" ; n = "<<n<<endl;
    swap1(&m,&n); // call by passing address of the variable
    cout<<"After swapping m = "<<m<<" ; n = "<<n<<endl<<endl;

    cout<<"Enter two integer : ";
    cin>>r>>s;
    cout<<"Before swapping r = "<<r<<" ; s = "<<s<<endl;
    swap2(r,s);
    cout<<"After swapping r = "<<r<<" ; s = "<<s<<endl;

    return 0;
}

```

```

Enter two integer : 20 10
Before swapping m = 20 ; n = 10
After swapping m = 10 ; n = 20

Enter two integer : 4 8
Before swapping r = 4 ; s = 8
After swapping r = 8 ; s = 4

Process returned 0 (0x0)   execution time : 11.813 s
Press any key to continue.

```

Q:6:: What is inline function? When will you make a function inline? Situation where inline function does not work?

Answer :

Inline function : An inline function is a function that is expanded inline when it is invoked, that is the compiler replaces the function call with the corresponding function code. The inline function are defined as follows :

```
inline      function_return_type      function_name(parameter_list)
{
    function body;
}
```

Example :

```
inline float cube(float a)
{
    return (a*a*a);
}
```

It is easy to make a function inline. All we need to do is to prefix the keyword inline to the function definition. All inline functions must be defined before they are called.

Situation when we can make a function inline : When the function definition size is small then to eliminate the cost of calls to that small function i.e. to get speed benefit the inline function can be used.

Because every time a function call takes a lot of extra time in executing a series of instructions for tasks such as jumping to the function, saving registers, pushing arguments into the stack, and returning to the calling function. When a function is small, a substantial percentage of execution time may be spent in such overheads.

Situation where inline function does not work :

- When function definition size is not small. The inline keyword merely sends a request not a command to the compiler. The compiler may ignore this request if the function definition is too long or too complicated and compile the function as a normal function.
- If a function contains a loop. (*for, while and do-while*)

- If a function contains static variables.
- If a function is recursive.
- If a function return type is other than void, and the return statement doesn't exist in a function body.
- If a function contains a switch or goto statement.

Q:7:: Write the advantages and disadvantages of inline function.

Answer:

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.

Disadvantages:

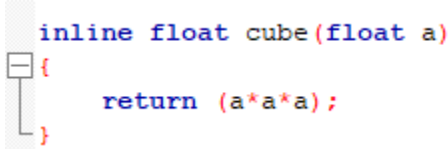
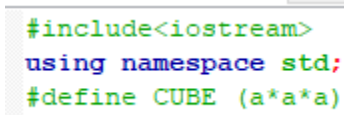
1. May increase function size so that it may not fit on the cache, causing lots of cache misses.
2. After inlining a function , if the number which are going to use the register increases then they may create overhead on register variable resource utilization.
3. It may cause compilation overhead as if some body changes code inside an inline function then all calling locations will also be compiled.
4. If it is used in a header file, it will make your header file size large and may also make it unreadable.

5. If somebody used too many inline functions resulting in a larger code size then it may cause thrashing in memory. More and more page faults bring down the program performance.

6. It's not useful for embedded systems where large binary size is not preferred at all due to memory size constraints.

Q:8:: How does inline function differ from preprocessor macro?

Answer :

	Inline function	Macro
1.	An inline function is defined by the "inline" keyword.	Whereas the macros are defined by the "#define" keyword.
2.	Through an inline function, the class's data members can be accessed.	Whereas macro can't access the class's data members.
3.	In the case of inline, the arguments are evaluated only once.	Whereas in the case of macro, the arguments are evaluated every time whenever macro is used in the program.
4.	In C++, inline may be defined either inside the class or outside the class.	Whereas the macro is all the time defined at the beginning of the program.
5.	In C++, inside the class, the short length functions are automatically made the inline functions.	While the macro is specifically defined.
6.	Inline is not as widely used as macros.	While the macro is widely used.
7.	Inline function is terminated by the curly brace at the end.	While the macro is not terminated by any symbol, it is terminated by a new line.
8.	 <pre>inline float cube(float a) { return (a*a*a); }</pre>	 <pre>#include<iostream> using namespace std; #define CUBE (a*a*a)</pre>

Q:9:: What do you mean by function overloading? When do we need function overloading-explain with an example.

Or,write a program in C++ that finds the area of different shapes using function overloading.

Answer :

Function overloading : Overloading refers to the use of the same thing for different purposes. Function overloading means we can use the same function name to create multiple functions and those functions can perform a variety of different tasks. This is known as function polymorphism in OOP.

Situation when we need function overloading : The function overloading in the c++ feature is used to improve the readability of the code. It is used so that the programmer does not have to remember various function names. Programmer can design a family of functions with one function name but with different argument lists. The function would perform different operations depending on the argument list in the function call. The correct function to be invoked is determined by checking the number and type of the arguments but not on the function type.

Here is an example program that finds the area of different shapes using function overloading:

Github code:

https://github.com/SyedaJannatul/DIIT_Object-Oriented-Programming/blob/9718bb46e4959dcfe6ad639c88685e659c8895ed/chapter%204_functions%20in%20C%2B%2B/chapter%204_class/question_9_code.cpp

```

#include<iostream>
using namespace std;

int area(int side)           //area of square
{
    return (side*side);
}

int area(int lenght, int width) //area of rectangle
{
    return (lenght*width);
}

float area(float radius)      //area of circle
{
    return (3.1416*radius*radius);
}

int main()
{
    int s,l,w;
    float r;
    cout<<"Enter the side of the square : ";
    cin>>s;
    cout<<"Area of the square : "<<area(s)<<endl<<endl;

    cout<<"Enter the length and width of the rectangle : ";
    cin>>l>>w;
    cout<<"Area of the rectangle : "<<area(l,w)<<endl<<endl;

    cout<<"Enter the radius of the circle : ";
    cin>>r;
    cout<<"Area of the circle : "<<area(r)<<endl<<endl;

    return 0;
}

```

```

Enter the side of the square : 5
Area of the square : 25

Enter the length and width of the rectangle : 5 10
Area of the rectangle : 50

Enter the radius of the circle : 5.5
Area of the circle : 95.0334

Process returned 0 (0x0)   execution time : 57.639 s
Press any key to continue.

```

Q:10:: Differentiate between function overloading and function overriding.

Answer :

	Function overloading	Function overriding
1.	It can not be used in the inheritance concept.	It is used in the inheritance concept.
2.	Though the functions name are same, the parameters type must be different.	Here the functions name are same and the parameters type must be same.
3.	It occurs within the same class or the program without using class concepts.	It occurs between two classes-derived class and base class.
4.	The functions return type may or may not be same.	The functions return type must be same.
5.	Here one function does not hide another function.	Here the derived class function hides the base class function.

Q: 11:: Write a c++ program that inputs a string from the keyboard and determine the length of that string.

Answer :

Github code :

https://github.com/SyedaJannatul/DIIT_Object-Oriented-Programming/blob/bfa30198bb1c529ffc7fc3357cfa375df8e84b7e/chapter%204_functions%20in%20C%2B%2B/chapter%204_class/question_11_code.cpp

```

#include<iostream>
#include<string.h>
using namespace std;

int strlenght(string s)
{
    int i;
    for (i = 0; s[i] ;i++)
        ;
    return i;
}

int main()
{
    string st;
    cout<<"Enter a string :";
    getline(cin,st);
    /*cin considers a space (whitespace, tabs, etc)
    as a terminating character, which means that
    it can only display a single word
    (even if you type many words).
    the getline() function to read a line of text.
    It takes cin as the first parameter,
    and the string object as second*/

    cout<<"String length : "<<endl;
    cout<<"\t Without using any built-in function : "<<strlenght(st)<<endl;
    cout<<"\t Using size() function : "<<st.size()<<endl;
    cout<<"\t Using length() function : "<<st.length()<<endl;
    cout<<"\t Using strlen() function : "<<strlen(st.c_str())<<endl;
    //c_str() convert the input object st into a character type array

    return 0;
}

```

```

Enter a string :hello c++ programming world!!
String length :
    Without using any built-in function : 29
    Using size() function : 29
    Using length() function : 29
    Using strlen() function : 29

Process returned 0 (0x0)   execution time : 76.063 s
Press any key to continue.

```


Q:12:: Write a c++ program that evaluate the following function to 0.0001% accuracy :

$$sum = 1 + (1/2)^2 + (1/3)^2 + (1/4)^2 + \dots\dots\dots$$

Answer :

Here, general term = $(1/n)^2$

Given accuracy = 0.0001%

So, here n will be:

$$\begin{aligned} \left(\frac{1}{n}\right)^2 &= 0.0001 \\ \Rightarrow \frac{1}{n} &= \sqrt{.0001} \\ \Rightarrow \frac{1}{n} &= 0.01 \\ \Rightarrow \frac{1}{n} &= \frac{1}{100} \\ n &= 100 \end{aligned}$$

That means, we have to calculate the sum of the first 100 terms of the series.

Github code -

https://github.com/SyedaJannatul/DIIT_Object-Oriented-Programming/blob/bfa30198bb1c529ffc7fc3357cfa375df8e84b7e/chapter%204_functions%20in%20C%2B%2B/chapter%204_class/question_12_code.cpp

```

#include<iostream>
#include<math.h>
#include <iostream>
#include <iomanip>
#define accuracy 0.0001
using namespace std;

double series()
{
    int i;
    double f,temp = 0.0,sum = 0.0,p = 2.0;

    for (i = 1; ; i++)
    {
        f = 1.0/i;
        temp = pow(f,p); //double pow(double,double);
        sum = sum + temp;
        if(temp<=accuracy)
            break;
        temp = 0.0;
    }
    cout<<"No. of terms n = "<<i<<endl;
    return sum;
}

int main()
{
    double s;
    s = series();
    cout<<"Sum = ";
    std::cout << std::fixed << std::setprecision(6) << s;
    return 0;
}

```

```

No. of terms n = 100
Sum = 1.634984
Process returned 0 (0x0)   execution time : 0.078 s
Press any key to continue.

```