Function Overloading, Copy Constructors, and Default Argument

Function Overloading

Function Overloading is the process of using same name for two or more than functions.

You can overload a function by using different data types of arguments or different number of arguments.

Note:- Two functions varying only in return type cannot be overloaded. For e.g. :- int num(int i) and float num(int i). These two functions cannot be overloaded as this is not possible in C++.

Sometimes two function appear to be different but in actual they are same. For e.g. :- int fun(int *a) and int fun(int a[]) .

Overloading of Constructor function is possible.

sscanf() :- The `sscanf()` function reads the data reads the data from buffer and stores the values into the respective variables. For eg:-

```
#include <cstdio>

int main ()
{
    char fname[50], lname[50];
    char buffer[] = "Kushagra/Shekhawat";

    sscanf(buffer, "%s%*c%s", fname, lname);
    printf("First name : %s \nLast name : %s\n", fname, lname);

    return 0;
}
```

sscanf() Parameters

- `buffer`: Pointer to a null-terminated character string to read the data from.
- `format`: Pointer to a null-terminated character string that specifies how to read the input. It consists of format specifiers starting with %.

  The format string has the following parts:

  - `Non whitespace characters` except % each of which consumes one identical character from the input stream. It can cause the function to fail if the next character on the stream does not compare equal.

- `Whitespace character`: All the consecutive whitespace characters are treated as single whitespace character. Further, '\n', '\t' and ' ' are considered same.
- `Conversion specification`: It follows the following format:
  - Initial % character that specifies the beginning
  - An optional * called assignment-suppressing character. If this character is present, fscanf() does not assign the result to any receiving argument.
  - An optional positive integer number that specifies maximum field width. It specifies the maximum number of characters that fscanf() is allowed to consume when doing the conversion specified by the current conversion specification.
  - An optional length modifier specifying the size of the receiving argument.
  - A conversion format specifier.

There are many problems faced when a object is used to initialize another object. For e.g. :-

When an object is passed as an argument in a function i.e. int fun(class_name obj)→

This will create a copy of the object(passed as an argument) using bitwise copy, This means obj will be exact copy of the object.

This means if memory was allocated to the object same memory will be shared by obj. Which means when we leave the function memory allocated to the obj will be destroyed, so will be the memory allocated to object, as they have same memory address.

To overcome this problem copy constructor was created.

See Program 14.1

Class_name obj1(12);

//……

Class_name obj2(12);

obj2 = obj1;

Here copy constructor will not be called. //This is related to Program. see Program

Here copy of obj1 is created by bitwise copy.

Address of an Overloaded Function

How to get Address of a function? See Program

Note :- Address of Overloaded and simple function are got by same way.

//See Program14.2

Function Overloading, Copy Constructors, and Default Argument

Default Function Arguments

In C++, it is possible to have a default function argument just as in Python.

The Default value will be taken by the parameter if no argument is passed, on the time of function calling.

For E.g. :- int fun(int d = 1);

The syntax is:-

Specify the default argument value in prototype, not in definition of the function.

See program 14.3 for details.

What is use of default argument?

- We can use it to minimize the number of function in case of function overloading
- We do not have to pass the same values again and again.
- Reduces the complexity of the code.

See program 14.3.1

Note:- The default parameter should only be specified in rightmost of the parameters, not in-between or in left. For e.g.:-

int func(int a=0, int b, int c), int func(int a, int b=0, int c) are incorrect

int func(int a, int b=0, int c=0), int func(int a, int b, int c=0) are correct.

Note :- Default argument should be used carefully as it can cause catastrophe.

Function Overloading and Ambiguity

The main cause of Ambiguity in C++ is due to automatic type conversion in function calls.

See program 14.4 and 14.4.1