

Function Overloading Implementation in C++ A Static Type of Polymorphism

Vishal V. Mehtre¹, Harshdeep²

¹Assistant Professor, Department of Electrical, Engineering Bharati Vidyapeeth Deemed University
College of Engineering, Pune, India

²Department of Electrical Engineering, Bharati Vidyapeeth Deemed University, College of Engineering,
Pune, India

Abstract- In this article the function overloading in object oriented programming is elaborated and how they are implemented in C++. The language supports a variety of programming styles. Here we are describing the polymorphism and its types in brief. The main stress is given on the function overloading implementation styles in the language. The polymorphic nature of languages has advantages like that we can add new code without requiring changes to the other classes and interfaces (in Java language) are easily implemented. In this technique, the run-time overhead is also introduced in dynamic binding which increases the execution time of the software. But there are no such types of overheads in this static type of polymorphism because everything is resolved at the time of compile time. **Polymorphism; Function Overloading; Static Polymorphism; Overloading; Compile Time Polymorphism**

Index terms- Polymorphism, Overloading, Static, Compile time

INTRODUCTION

The word polymorphism is used in coding. It has the origin from Ancient Greek words. Here Poly means many and morph means form. Basically it is the ability to assume several different forms. It is used in Chemistry as the same. In Microbiology (the branch of biology that studies microorganisms and their effects on humans), polymorphism word is used, it means the ability of some bacteria to alter their shape or size in response to environmental conditions.

In Cytology (a branch of Biology concerned with the structure and function of plant and animal cells), polymorphism means variability in the size and shape of cells and/or their nuclei. The taxonomy of the function overloading is shown in the Fig. 1.

STATIC POLYMORPHISM

Function Overloading

C++ allows specification of more than one function of the same name in the same scope. These are called overloaded functions and are described in detail in Overloading. Overloaded functions enable programmers to supply different semantics for a function, depending on the types and number of arguments.

Operator Overloading

It is ad-hoc type of polymorphism. It allows the programmer to redefine how standard operators work with class objects (Class - user-defined data type). [1] We always consider these two things – first is the overloaded operators for classes should behave like operators for built-in types and the overloaded operators should have the same semantics as their built-in counterparts.

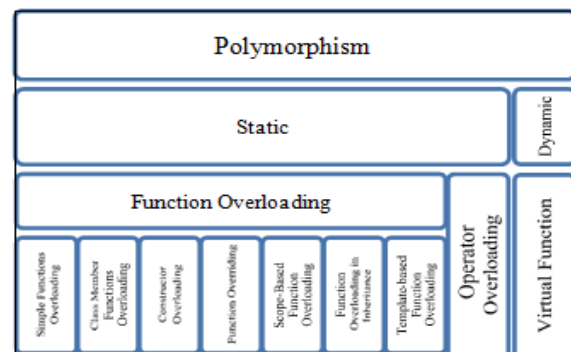


Fig. 1. Taxonomy of function overloading

DYNAMIC POLYMORPHISM

The polymorphism exhibited at run time is called dynamic (run- time determination) polymorphism.

What makes more sense is to use run-time binding where the decision about which version of the method to invoke is delayed until runtime.

In case of static polymorphism, the compiler resolves which function to call at the compile time. But in the case of dynamic polymorphism, this decision is delayed until the runtime.

Virtual Function

It is a dynamic polymorphism a virtual function is a member function that you expect to be redefined in derived classes. When you refer to a derived class object using a pointer or a reference to the base class, you can call a virtual function for that object and execute the derived class's version of the function.

Virtual functions ensure that the correct function is called for an object, regardless of the expression used to make the function call. A virtual function is a function in a base class that forms part of the interface for a set of derived classes.

It is declared virtual in the base class and may or may not have a definition in that class. It will have definitions in one or more of the derived classes.[2]

The purpose of a virtual function is to have one name, one prototype, and more than one definition so that the function's behavior can be appropriate for each of the derived classes.

FUNCTION OVERLOADING

In it the same variable name is used to denote different functions, and the context is used to decide which function is denoted by a particular instance of the name. Function (Method) overloading Rules:

1. All the overloading functions must have the same function name.
2. The number of arguments may differ.
3. The data types of arguments may differ.
4. The order of arguments may differ.
5. The return type may differ. But only the return type difference is not counted in the function overloading. (see the note)

Note: Methods (or functions) with difference in only the return type of prototypes and same in both the number of arguments and as well as the data types of arguments, are treated same by the compiler because the compiler considers only the parameters. The following prototypes are treated same by the compiler. [5]

```
int sum (int a, float b); int sum (int a, int b); float sum (int a, int b);
```

Function overloading a method (function) allows you to create functions of the same name that take different data types of arguments or different in number of arguments or different in the sequence of arguments as follows:

- a. Methods (or functions) with difference in the data type of arguments of prototypes.
int sum (int a, int b);
- b. Methods (or functions) with difference in the number of arguments of prototypes.
int sum (int a, int b);
int sum (int a, int b, int c);
- c. Methods (or functions) with difference in the sequence of arguments of prototypes.
int sum (int a, float b);
int sum (float a, int a);

The function overloading is further explored in the following subsections.

Simple Function Overloading

Constructors are the special type of member functions of a class. These are called special because of the following properties: o A constructor's name is same as the name of the class, concerned.

- These do not have a return value. o These are called automatically when an object of the class is created.
- These are always declared in the public section of the class.
- These are used to initialize the area fields for the concerned objects.

Other properties are same as of the normal functions or member functions of a class have. So these are also overloaded as the other functions. [7]

Function Overriding

The function overriding takes place in inheritance (an OOPS concept). It allows replacing an inherited method with a different implementation under the same name (same prototype). [8] Usually the overridden function will have the same number, order, and types of arguments. Since, it is proposed to be a matching replacement.

Under this concept both the classes (base class and derived class) have member functions with same name and arguments under the public access specifier and inherited the base class publicly. If we create an

object of derived class and write code to access that member function then, the member function in derived class is only invoked. [3] Here, the member function of derived class overrides the member function of base class. It can be resolved with the help of scope resolution operator (: :), such as object.base::function_name (arg1, arg2, ...).

Scope-based Function Overloading

Some methods are implemented in different scope with the same prototype (Function Header). The scopes cannot overlap with each other. There semantics may be different but the same prototype. The type signature may be different. It is shown in the Fig. 2.

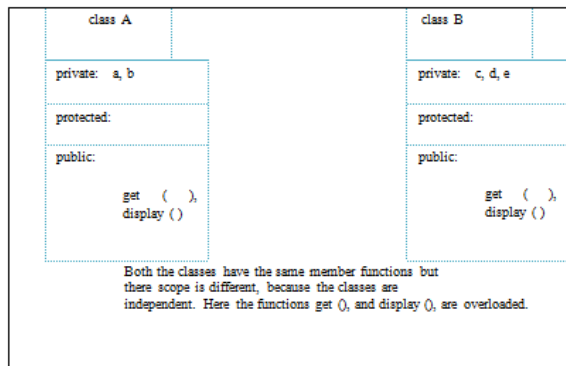


Fig.2. Scope-based Function overriding of two classes

Template-based Function Overloading

A template function may be overloaded either by template functions or ordinary functions of its name. [9] In such situations the overloading resolution is accomplished as follows:

- o Call an ordinary function that has an exact match.
- o Call a template function that could be created with an exact match.
- o Try normal overloading resolution to ordinary and call the one that matches.

IMPLEMENTATION

All the implementations were done on the compiler Microsoft Visual C++ 6.0. The operating system was Microsoft Windows XP Version 2002 Service Pack 3. The Primary memory (RAM) is of 2 GB. The processor is Intel(R) Core (TM) 2 Duo CPU E7400 @ 2.80GHz. All the programs are also run on the freely available online compiler C++ 4.7.2 (gcc-4.7.2) on ideone.com. It is an online compiler and debugging tool which allows us to compile and run

code online in more than 40 programming languages. The web browser, Google Chrome was used to access the website www.ideone.com.

CONCLUSION & FUTURE SCOPE

This is basically the empirical study of the function overloading, the C++ programming language concept. It is the static polymorphism which may have many variations in the implementation. This paper is good for the students and C++ language lovers to understand the function overloading deeply. In future works we are going to elaborate the dynamic polymorphism, a powerful mechanism in the object oriented programming languages like C++.

ACKNOWLEDGMENT

The authors are highly grateful to the anonymous reviewers who have commented on this paper. Their detailed and constructive feedback helps a lot in representing the contents and the presentation of the paper.

REFERENCES

- [1] M. Young, The Technical Writer's Handbook. Mill Valley, CA: University Science, 1989.
- [2] Owen Astrachan, "A Computer Science Tapestry: Exploring Computer Science with C++," McGraw- Hill Science. December 12, 2000.
- [3] Lennart Augustsson, "Implementing Haskell overloading," FPCA '93: Conference on Functional Programming Languages and Computer Architecture, Copenhagen, Denmark. ACM Press, pp. 341-349, June 1993.
- [4] E. Balagurusamy, "Object Oriented Programming with C++," Tata McGraw Hill Education Pvt. Ltd. 2011.
- [5] Med Maria Bleker, "About Pleomorphism," Explore Publication, Inc. 1998. http://www.pnf.org/compendium/About_Pleomorphism_Bleker.pdf
- [6] Luca Cardelli, and Peter Wegner, "On Understanding Types, Data Abstraction, and Polymorphism," Computing Surveys. Volume 17, Number 4, pp. 471-522, 1985.
- [6] Mark P. Jones, "Functional Programming with Overloading and Higher-order Polymorphism,"

- J. Jeuring and E. Meijer, editors, *Advanced Functional Programming*, LNCS Springer-Verlag, 925, pp. 97-136, 1995.
- [7] Poornachandra Sarang, "Object-Oriented Programming with C++," PHI, ISBN: 978-81-203-3670-4, 2006.
- [8] M. Shields and S. Peyton Jones, "Object-oriented style overloading for Haskell," BABEL'01: Workshop on MultiLanguage Infrastructure and Interoperability, pp. 1-20, September 2001.
- [9] Cheng Wang and Daqing Hou, "An Empirical Study of Function Overloading in C++," 8th IEEE International Working Conference on Source Code Analysis and Manipulation, pp. 4756, 2008.