**Overloaded functions sub:Oop’s with C++**

**Q what is function overloading?**

**Overloaded functions:**

Two or more functions with the same name declared in the same scope but with

different parameters are called overloaded functions. C++ supports a special kind of

functions called overloaded functions. The C++ supports the several functions with the

same name but with different sets of parameters is called function overloading. This is

known as function polymorphism in oops.

When an Overloaded function is invoked the C++ complier chooses the

appropriate function by examining the number, data type, and order of the arguments

present in that function call. Overloaded functions are used to perform similar tasks, but

on different data types.

Overloaded functions are distinguished by their respective names followed by the

parameter list in the function definition.

The function name and parameter list help in type safe linkage. The number of

parameters and their data types are useful in ensuring that the proper overloaded function

is invoked. In doing so, it also confirms the one-to-one correspondence between the

arguments and the parameter.

**Why ? To Over-load a function name:**

The main objective behind overloading the functions is to make programs more

readable and understandable, if they perform closely related tasks. For example, if you

want to display an int, a float and a string, then you would need three separate functions

to indicate the same function although with different data types. Probably these functions

may be displayInt( ) for displaying integers, displayfloat( ) for displaying floating point

numbers and displayString( ) for displaying string objects. Now think about the number

of function names that we need to be remembered and used while implementing them for

a very large set of data types. Then it would become not only cumbersome task but also

inefficient way from the software engineering point of view. So, this makes us to go for

writing the functions having the same name but with different parameters. By doing so

we can save memory and time.

How to overload a function name:

In C++, two or more functions can be given the same name provided that each

parameter list is unique in either the number or the types of parameters.

Ex 1: 1. int add (int ival1, int ival2);

2. float add (float fval1, float fval2);

3. double add(double dval1, double dval2, double dval3);

4. char add(char ch1, char ch2);

The above overloaded functions are all having their unique set of parameters. A

overloaded function can also contain the parameter list with mixed data types.

Ex 2: Declarations

int add(int a, int b); //Prototype 1

int add(int a, int b, int c); //Prototype 2

double add(double x, double y); //Prototype 3

double add(int p, double q); //Prototype 4

double add(double p, int q); //Prototype 5 Overloaded functions sub:Oop’s with C++

// Function calls

1. Cout<<add(5, 10); // Uses Prototype 1

2. Coot<<add(15, 10.0); // Uses Prototype 4

3. cout<<add(12.5, 7.5); // Uses Prototype 3

4. cout<<add(5, 10, 15); // Uses Prototype 2

5. cout<<add(0.75, 5); // Uses Prototype 5

A function call first matches the prototype having the same number and type of

arguments and then calls the appropriate function for execution. The complier also

distinguish between overloaded functions with the same number of arguments, provided

their type is different.

Example: For overloaded functions

# include<iostream.h>

# include<stdio.h>

# include<conio.h>

void add(3, 4);

void add(5.5, 7.3);

void add(2, 3, 4);

void add(2.5, 5);

void add(5,2.4);

void main ( ){

clrscr();

add(3, 4);

add(5.5, 7.3);

add(2, 3, 4);

add(2.5, 5);

add(5,2.4);

getch();

}

void add(int i, int j){

cout<<”sum = “<<i+j<<endl;

cout<< endl;

}

void add(double i, double j){

cout<<”sum = “<<i+j<<endl;

cout<< endl;

} Overloaded functions sub:Oop’s with C++

void add(int i, int j, int k){

cout<<”sum = “<<i+j+k<<endl;

cout<< endl;

}

void add(double i, int j){

cout<<”sum = “<<i+j<<endl;

cout<< endl;

}

void add(int i, double j){

cout<<”sum = “<<i+j<<endl;

cout<< endl;

}

When a function name is declared more than once in a particular scope, the

compiler interprets the declarations as follows:

• The parameter lists of two or more functions may differ in either the number of

parameters or their data types. In such cases, the compiler interprets these

functions as overloaded functions.

Example:

int sum(int x, int y);

float sum(float a, float b, float x);

• Two or more functions declarations may have exactly the same return type and

the parameter list. If such is the case, the compiler will generate- “Possibly

redeclaration of function….” error for the second function declaration.

Example:

int sum(int x,int y);

int sum(int x,int y); // redeclaration error.

• If the parameter lists of the two functions match exactly but the return types

differ, the second declaration is treated as an erroneous redeclaration of the first

and is flagged at compile –time as an error. Overloaded functions sub:Oop’s with C++

Example:

int sum(int x, int y);

unsigned int sum(int x, int y); //erroneous redeclaration.

• Two or more function declaration may differ only in their default arguments. In

such case, the compiler will cause “redeclaration of the first” error.

Example:

int volume(int length, int breadth, int height);

int volume(int ,int, int=20); //redeclaration of first.

• The following program segment would give an error

typedef INTR int ;

void display(int);

void display(INTR);

The example above would not compile correctly because the compiler has no way

of differentiating between the two versions of the function display( ).

An INTR is just another name for an int, typedef just gives another name for an

existing type and does not constitute an original type of its own.

• Which function gets called in the following example?

void f();

void f(int);

void f(double , double=3.4);

void f(char \*, char \*);

void main(){

f(5.6);

return 0;

}

• The following declarations declare the same function.

void f(int);

void f(const int);

The function f can be defined as

void f(int)

{ }

or as

void f(const int)

{ }

Providing both of these definitions in the same program is an error, however,

because these definitions define the same function twice. Overloaded functions sub:Oop’s with C++

• Do you think that the following function would work?

#include<iostream.h>

void display(char \*);

void display(const char\*);

void main( ){

char \*ch1=”Hello”

const char \*ch2=”Bye”

display(ch1);

display(ch2);

}

void display(char \*p3){

cout<<p3;

}

void display(const char \*p){

cout<<p;

}

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*End of overloadingfunctions\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*