Polymorphism-I

Example

- We have types of employees in Govt. Org.
- Regular, Daily wages and Adhoc.
- Gross Pay for the employees are calculated as follows:
  - Regular employees - Basic + HRA + DA + TA
  - Daily wages -wages per hour * number of hours
  - Adhoc- fixed amount

Finding Gross pay

Input	Output	Calculation?
Components for calculating gross pay	Gross pay	Calculation based on type of employees

How do we write functions?

- Same function name for all three type of employees
- More meaningful and elegant way of doing things
- Calculate_Gross_Pay for all types of employees

Polymorphism

- Refers to ‘one name having many forms’, ‘one interface doing multiple actions’.
- In C++, polymorphism can be either
  - Compile time or static time polymorphism or
  - Run time or dynamic time polymorphism
- C++ implements *static polymorphism* through
  - *overloaded functions*
  - *overloaded operators*

Polymorphism

- Greek word- many forms
- Single name can be used for different purposes
- How is polymorphism done?
  - Function overloading
  - Operator overloading
  - Dynamic binding

Overloading

- Overloading – Two or more different meanings for the name
- Overloaded function - a function having more than one distinct meanings
- Overloaded operator - When two or more distinct meanings are defined for an operator

Overloading Operators

- C and C++ have operator overloading inbuilt.
- ‘-’ : unary and binary
- ‘*’ :multiplication and pointers
- ‘<<‘, ‘>>’ : bitwise shift as well as insertion and extraction operators
- All arithmetic operators can work with any type of data

Function Overloading

- Several functions of the same name can be defined, as long as they have different signatures.

- The C++ compiler selects an appropriate function to call by considering the number, types and order of the arguments in the call.

Function Overloading Cont..

- Overloaded functions are distinguished by their signatures
- Signature - Combination of a function's name and its parameter types (in order)
- Internally, C++ compilers does encode each function identifier with the number and types of its parameters referred to as name mangling or name decoration to enable type-safe linkage.

Example

```
#include <iostream>
using namespace std;
int square (int y) {
return y *y;}
double square(double y) {
return y *y;}
int main( )
{
cout << square(10) << endl;
cout << square(10) << endl;
}
```

Function's Signature

- A function's argument list (i.e., number and type of arguments) is known as the function's signature.

- Functions with same signature - Two functions with same number and types of arguments in same order, variable names doesn't matter.

- Following two functions have same signature.

```
void square (int x, float y);
void square (int c, float d);
```

Some examples

- void print (int i);
- void print (char c);
- void print(float f);
- void print(double d);

Resolution by Compiler

- Signature of subsequent functions match previous function's, then the second is treated as a re-declaration of the first - Error
- Signatures of two functions match exactly but the return type differ, then the second declaration is treated as an erroneous re-declaration of the first
- For example,

```
float square (float f);
double square (float x);
// Differ only by return type so erroneous re-declaration
```

- If the signature of the two functions differ in either the number or type of their arguments, the two functions are considered to be overloaded.

Finding a Match

- To resolve a particular instance of the function, following there cases can be considered.
- A match is found for the function call.
- A match is not found for the function call.
- There is more than one defined instance for the function call.

Function Match

- *void* show (int);
- *void show* (double);
- The function call
- *show* (0);
- Matched to *void show* (int); and compiler invokes corresponding function definition **as 0** (zero) is of type **int**

Promotion

- Promotion of the actual argument if no exact match is found.
- The conversion of integer types **char, short** into **int** - *integral promotion.*
- For example, consider the following code fragment:

  void show (int);

  void show (float);

  show (‘*c*’);

- What will be invoked?

Example

```
#include <iostream>
using namespace std;
void show (int i)
{
cout << "I am interger" << i << endl;

}
void show(float f)
{
 cout << "I am float " << f << endl;
}
int main()
{
 //case 2 - treated as int .. a match through promotion
show('c'); // converted to int and displayed

}
```

Example

```
#include <iostream>
using namespace std;
void show (int i)
{
cout << "I am interger" << i << endl;

}
void show (char c)
{
cout << "I am character " <<c<< endl;
}
```

```
int main()
{
 //case 2 - treated as char  ..

show('c');  // Used as char

}
```

Application of standard C++ conversion rules

- If we do not have a direct/exact match
- If we do not have a match through a promotion
- Try to get a match through a standard conversion of the actual argument.
- Example:

```
void show(char);
show(97);
```

- A match through standard conversion matches show(char)

Example

```
#include <iostream>
using namespace std;
void show (char c)
{
cout << "I am character " <<c<< endl;
}

int main()
{
  show(97); // int to char
}
```

```
#include <iostream>
using namespace std;
void show (int i)
{
cout << "I am interger" << i << endl;

}
int main()
{
    show(97.5F); // float to int
 }
```

Example -Ambiguous Call

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

void show (double i)
{
        cout << "I am interger" << i << endl;

}
void show (char c)
{
        cout << "I am character " <<c<< endl;

}
```

```
int main()
{

  show(97); // Ambiguous call

}
```

Default Arguments Versus Overloading

- Using default arguments in functions is also overloading, because the function may be called with an optional number of arguments.

- For instance, consider the following function prototype:
- float cal_amount(float principal,int time=5,float rate=0.12);

- Function Calls
- cout<<cal_amount(5000);
- cout <<cal_amount (5000,5);
- cout <<cal_amount (5000,5,0.18);

Example

```
#include <iostream>
using namespace std;
double calc_gross_pay(float, float, float);
double calc_gross_pay(float, float);
double calc_gross_pay(float = 50000);

double calc_gross_pay(float basic, float da, float hra)
{       return basic + da/100 *basic + hra ; }
double calc_gross_pay(float hr, float wg)
{       return hr * wg ;    }
double calc_gross_pay(float pay)
{       return pay ; }
```

Example

```
int main()
{
int ch;
double gross;
cout<<"Enter choice" << endl;
cin>> ch;
if (ch==1)
{
float basic, DA, HRA;
cout<<"Enter basic, da, and hra";
cin>>basic>>DA>>HRA;
cout<<calc_gross_pay(basic, DA,
HRA);
}
else if (ch==2) {
float hrs, wages_hr;
cout<<"Enter hrs and wages per hour";
cin>>hrs>>wages_hr;
cout<<calc_gross_pay(hrs, wages_hr);
else if (ch==3)
{ cout<<calc_gross_pay(); }
else if (ch==4){
float pay;
cout<<"Enter new consolidated salary";
cin >> pay;
cout<<calc_gross_pay(pay);
}
}
```

References

- C++: The Complete Reference, 4th Edition by Herbert Schildt , McGraw-Hill
- Teach Yourself C++ 3rd Edition by Herbert Schildt,
- The C+ + Programming Language, Third Edition by Bjarne Stroustrup, Addison Wesley