Function Overloading

Lecture 12

Outline

1. Function Overloading
2. Rules and Gotchas

Function Overloading

- C++ allows you to declare multiple functions with the same name under certain conditions
  - Each declaration must have different arguments
    - Different number and/or types of arguments

- This is called *function overloading*

- Most useful when you need multiple functions that serve a similar purpose

Example (number)

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

double max(double num1, double num2);
double max(double num1, double num2, double num3);

int main()
{
     cout << "Max of 5.5 and 6.7 is "
          << max( 5.5, 6.7 ) << endl;
     cout << "Max of 7.2, 1.9, and -4.5 is "
          << max( 7.2, 1.9, -4.5 ) << endl;
     return 0;
}

double max(double num1, double num2)
{
     if ( num1 > num2 )
     {
          return num1;
     }
     return num2;
}
```

```
double max(double num1,
           double num2, double num3)
{
     if ( num1 > num2 && num1 > num3 )
     {
          return num1;
     }
     else if ( num2 > num3 )
     {
          return num2;
     }
     return num3;
}
```

Both functions named **max**, but have different number of arguments

Example (type)

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

int do_calc(int x);
double do_calc(double x);

int main()
{
     cout << "do_calc( 5 ) = " << do_calc( 5 ) << endl;
     cout << "do_calc( 5.5 ) = " << do_calc( 5.5 ) << endl;
     return 0;
}

int do_calc(int x)
{
     return ( x*x );
}

double do_calc(double x)
{
     return ( x/10 );
}
```

Both functions named **do_calc**, but have arguments of different types

Rules and Gotchas

- C++ finds the matching function for a particular call by matching the number of arguments and the argument types
- Functions can't be overloaded when all that is different is the return type (has to have a different argument list)
- C++ always tries to find an exact match on arguments first, then tries automatic conversions (for example, from `int` to **double**)

Exercise

Write two functions named **`area`**. One should compute the area of a circle and so has a single argument, which is the radius. The second should compute the area of a rectangle and so has two arguments, which are the length and the width. Write a **`main`** function to test them both.

Answer

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

double area(double radius);
double area(double length, double width);

int main()
{
     cout << "Area of circle with radius 5 is: " << area( 5.0 ) << endl;
     cout << "Area of rectangle with sides 3.5 and 4 is: " << area( 3.5, 4 ) << endl;
     return 0;
}

double area(double radius)
{
     const double PI = 3.14159;
     return ( PI * radius * radius );
}

double area(double length, double width)
{
     return ( length * width );
}
```

Exercise

Write three functions named **`get_input`** that read a value from the user. One gets an **`int`** value from the user, one gets a **`double`** value, and one gets a **`char`** value. Note that just changing the return type won't be enough, so you should use call by reference arguments for each. Write a **`main`** function to test all three.

Answer

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

void get_input(int& input);
void get_input(double& input);
void get_input(char& input);

int main()
{
      int i;
      double d;
      char c;
      get_input( i );
      get_input( d );
      get_input( c );
      cout << "Read: " << i << ", ”
           << d << ", " << c << endl;
      return 0;
}

void get_input(int& input)
{
      cout << "Enter an integer: ";
      cin >> input;
}
```

```
void get_input(double& input)
{
      cout << "Enter a number: ";
      cin >> input;
}

void get_input(char& input)
{
      cout << "Enter a character: ";
      cin >> input;
}
```

Wrap Up

- Multiple functions can have the same name but different definitions
  - Called *function overloading*
- Functions must have different argument lists
- Mostly used when doing similar calculations with different types of arguments
- Can't overload functions only by changing the return type