

Object Oriented Programming

Concepts

Lecture 3

University of Anbar

College of Computer Science and Information Technology

Department of Computer Science

Object Oriented Programming

Second Class

Dr. Ruqayah R. Al-Dahhan

Outlines:

- Object Oriented Programming

Benefits

-Functions:

- overloading
- reference parameters

Object Oriented Programming

- An object is an abstract **data type** created by a user. It can include multiple properties and **methods** and may even contain other objects. In most **programming languages**, objects are defined as **classes**.
- Objects maybe represent a person, a place, a bank account, etc.
- The main principles of object-oriented programming are **encapsulation**, **inheritance**, and **polymorphism**.

Object Oriented Programming Benefits

- **Modularity**; object maintained independently of the other objects.
- **Information-hiding**; through methods, the details remain hidden.
- **Code re-use**; re-use of object in the new program.
- **Debugging ease**; easier to spot problematic parts of program.

Function Overloading

```
int max(int m, int n) { // max of two ints
    return (m>n)?m:n;
}
```

What about the max of two floats?

max is already in use -

```
float fmax(float x, float y) {
    return (x>y)?x:y;
}
```

But it adds to the clutter of our program to have a different name.

Function Overloading

A major topic in OOP is overloading methods, which lets you define the same method multiple times so that you can call them with different argument lists.

Function Overloading

C++ allows functions with arguments of differing types, to have the same name.

We call this **function** overloading.

```
int max(int m, int n) {  
    return (m>n)?m:n;  
}  
float max(float x, float y) { //is OK  
    return (x>y)?x:y;  
}
```

Function Overloading

An overloaded function must have parameters that differ in some way. Differing only by return type is not allowed. Otherwise the compiler would be unable to distinguish which version we mean because of the type promotion rules.

```
int triple(int a) {  
    return 3*a;  
}
```

```
float triple(int a) { //Not allowed!!  
    return 3.0*a;  
}
```


Function Overloading

- One function can have more arguments than another, or the types of the arguments can be different.
- The compiler chooses which function to use by matching the arguments.

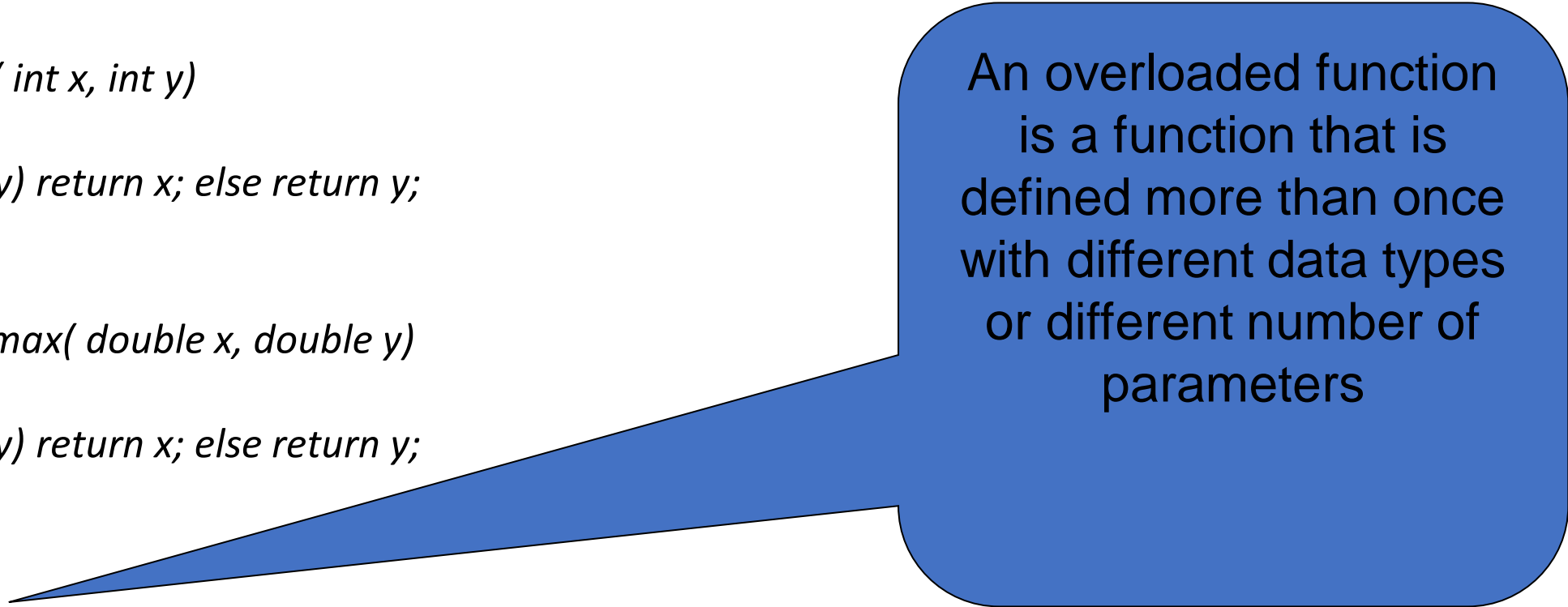
Example

Function Overloading

- Write functions to return with the maximum number of two numbers

```
int max( int x, int y)  
{  
    if (x>y) return x; else return y;  
}
```

```
double max( double x, double y)  
{  
    if (x>y) return x; else return y;  
}
```



An overloaded function is a function that is defined more than once with different data types or different number of parameters

Call-by-reference

In C++, functions can use **call-by-reference**. (A different use of the ampersand & symbol - here it says this argument is call-by-reference) In the following the printout is **j = 6**

```
int main() {
    int j = 4;
    changeIt(j); // j will be passed by ref
    normal(j); // j passed as a copy
    cout << "j = " << j << endl; // j is now 6
}

void changeIt(int &p) {
    p = p + 2;
}

void normal( int p ){
    p =7; // only acts on passed copy
}
```

Call-by-reference

p is called '**reference parameter**'. It refers back to the original variable - so the function can alter its parameter.

```
void changeIt(int &p) {  
    p = p + 2;  
}
```

Call-by-reference

We can create 'reference variables' similarly:

```
int main() {  
    int m;  
    int &q = m;  
    ...  
}
```

q is now just another name for the variable m.
And we can use it to manipulate the actual variable m.

Summary

Overload - to specify more than one function of the same name, but with varying numbers and types of parameters.

Reference - another name for a variable. Access to a variable via a reference is like manipulating the variable itself.