Functions

Mohamed Saied

Function Definition

Define *function header* and *function body*

Value-returning functions

- return-data-type function-name(parameter list)
- {
- constant declarations
- variable declarations
- other C++ statements
- return value
- }

Function Definition (cont.)

- *Non value-returning* functions

```
void function-name(parameter list)
{
 constant declarations
 variable declarations

 other C++ statements
}
```

Function Definition (cont.)

- The argument names in the function header are referred to as *formal parameters*.

```
int FindMax(int x, int y)
{
 int maximum;

 if(x>=y)
   maximum = x;
 else
   maximum = y;

 return maximum;
}
```

Function Prototype

- Every function should have a *function prototype*.
- The function prototype specifies the *type* of the value that the function returns (if any) and the *type*, *number*, and *order* of the function's arguments.

return-data-type function-name(argument data types);

or

void function-name(argument data types);

Function Prototype (cont.)

The use of function prototypes permits *error checking* of data types by the compiler.

It also ensures conversion of all arguments passed to the function to the declared argument data type when the function is called.

Preconditions and Postconditions

Preconditions are a set of conditions required by a function to be true if it is to operate correctly.

Postconditions are a set of conditions required to be true after the function is executed, assuming that the preconditions are met.

Preconditions and Postconditions (cont.)

```
int leapyr(int)

// Preconditions: the integers must represent a year in // a
four digit form, such as 1999

// Postconditions: a 1 will be returned if the year is a  // leap
year; otherwise, a 0 will be returned

{

C++ code

}
```

Calling a function

A function is *called* by specifying its name followed by its arguments.

Non-value returning functions:

- *function-name (data passed to function);*

Value returning functions:

- *results = function-name (data passed to function);*

Calling a function (cont.)

```
#include <iostream.h>

int FindMax(int, int); // function prototype

int main()
{
 int firstnum, secnum, max;

 cout << "\nEnter two numbers: ";
 cin >> firstnum >> secnum;

 max=FindMax(firstnum, secnum); // the function is called here

 cout << "The maximum is " << max << endl;

 return 0;
}
```

- The argument names in the function call are referred to as *actual parameters*

Calling a function by value

The function receives a copy of the actual parameter values.

The function *cannot* change the values of the actual parameters.

Calling a function by reference

Very useful when we need a function which "returns more than one value".

The formal parameter becomes an alias for the actual parameter.

The function *can* change the values of the actual parameters.

Calling a function by reference (cont.)

```
#include <iostream.h>
void newval(float&, float&); // function prototype
int main()
{
float firstnum, secnum;
cout << "Enter two numbers: ";
cin >> firstnum >> secnum;
newval(firstnum, secnum);
cout << firstnum << secnum << endl;
return 0;
}
void newval(float& xnum, float& ynum)
{
xnum = 89.5;
ynum = 99.5;
}
```

The "const" modifier

Call by reference is the *preferred* way to pass a large structure or class instances to functions, since the entire structure need not be copied each time it is used!!

C++ provides us with protection against accidentally changing the values of variables passed by reference with the *const* operator

- function prototype: *int FindMax(const int&, const int&);*
- function header: *int FindMax(const int& x, const int& y);*

Function Overloading

C++ provides the capability of using the same function name for more than one function (*function overloading*).

The compiler must be able to determine which function to use based on the number and data types of the parameters.

Function Overloading (cont.)

```
void cdabs(int x)
{
 if (x<0)
   x = -x;
 cout << "The abs value of the integer is " << x << endl;
}

void cdabs(float x)
{
 if (x<0)
   x = -x;
 cout << "The abs value of the float is " << x << endl;
}
```

- *Warning*: creating overloaded functions with identical parameter lists and different return types is a syntax error !!