CSE 1325

Week of 11/21/2022

Instructor : Donna French

Command Line Arguments

Passing command line arguments

Executable programs can be run on the command line by invoking them by name.

```
./Code1_1000074079.e
```

In order to pass command line arguments to a program, we list the command line arguments after the executable name

```
./Code1_1000074079.e FileToRead.txt
```

Command Line Parameters

Running a program with command line parameters

```
./Code1_1000074079.e clp1 clp2 clp3
```

Running a program in debug with command line parameters

```
gdb --args ./Code1_1000074079.e clp1 clp2 clp3
```

```
#include <iostream>
#include <vector>

int main(int argc, char *argv[])
{
    int i;
    std::vector<std::string>CatNames{};

    for (i = 1; i < argc; i++)
    {
        CatNames.push_back(argv[i]);
    }

    for (auto it : CatNames)
        std::cout << it << "\t";

    return 0;
}
```

Default Arguments

A **default argument** is a default value provided for a function parameter.

If the user does not supply an explicit argument for a parameter with a default argument, the default value will be used.

If the user does supply an argument for the parameter, the user-supplied argument is used.

Default Arguments

If a function is repeatedly invoked with the same argument value for a particular parameter, then

you can specify that such a parameter has a default argument

Default argument – a default value is passed to that parameter

When a program omits an argument for a parameter with a default argument in a function call, the compiler rewrites the function call and inserts the default value of that argument.

```
unsigned int boxVolume(unsigned int length=1, unsigned int width=1, unsigned int height=1)
{
        return length * width * height;
}

int main()
{
        // nothing is passed - use defaults for all
        cout << "boxVolume() = " << boxVolume() << endl;

        // length is passed - use default width and height
        cout << "\n\nboxVolume(10) = " << boxVolume(10) << endl;

        // length and width are passed - use default height
        cout << "\n\nboxVolume(10,5) = " << boxVolume(10,5) << endl;

        // length and width and height are all passed - no defaults
        cout << "\n\nboxVolume(10,5,2) = " << boxVolume(10,5,2) << endl;

        return 0;
}
```

default values

Default Arguments

Default values need to be specified in EITHER the prototype or the function **BUT** not both.

```
unsigned int boxVolume(unsigned int length, unsigned int width, unsigned int height);

unsigned int boxVolume(unsigned int length=1, unsigned int width=1, unsigned int height=1)
{
        return length * width * height;
}
```

OR

```
unsigned int boxVolume(unsigned int length = 1, unsigned int width = 1, unsigned int height = 1);

unsigned int boxVolume(unsigned int length, unsigned int width, unsigned int height)
{
        return length * width * height;
}
```

Preferred method – just makes the defaults easier to find in the program

Default Arguments

```
#include <iostream>

using namespace std;

void PrintIT(string Word3, string Word1="University of Texas", string Word2=" at ");

void PrintIT(string Word3, string Word1, string Word2)
{
   cout << Word1 << Word2 << Word3 << endl;
}

int main()
{
   PrintIT("Arlington");
   PrintIT("Austin");
   PrintIT("Dallas");

   return 0;
}
```

```
University of Texas at Arlington
University of Texas at Austin
University of Texas at Dallas
```

Default Arguments

```
University of Texas at Arlington
University of Texas at Austin
University of Texas at Dallas
```

```
#include <iostream>

using namespace std;

void PrintIT(string Word3, string Word1="University of Texas", string Word2=" at ");

void PrintIT(string Word3, string Word1, string Word2)
{
   cout << Word1 << Word2 << Word3 << endl;
}

int main()
{
   PrintIT("Arlington");
   PrintIT("Austin");
   PrintIT("Dallas");

   return 0;
}
```

How to print

```
University of Tulsa
PrintIT("", "University of Tulsa", "");
Texas A&M University-Commerce
PrintIT("Commerce", "Texas A&M University", "-");
Texas A&M University at Galveston
PrintIT("Galveston", "Texas A&M University");
```

Function Overloading

Function overloading is a feature of C++ that allows us to create multiple functions with the same name, so long as they have different parameters.

The C++ compiler selects the proper function to call by examining the number, types and order of the arguments in the call.

The combination of a function's name and its parameters types and the order of them is called a **signature**.

`this`

Recommendation

Do not add `this->` to all uses of your class members.

Only do so when you have a specific reason to.

Operator Overloading

>>

stream extraction operator

bitwise right shift operator

<<

stream insertion operator

bitwise left shift operator

These are familiar operators that are overloaded.

Operator Overloading

+ and –

Each of these performs differently depending on their context

integer addition
floating point arithmetic
pointer arithmetic

These are familiar operators that are overloaded meaning that the compiler generates the appropriate code based on the types of the operands.

Operator Overloading

Most of C++'s operators can be overloaded.

There are a few exceptions

- `.`
- `.*` (pointer to member)
- `::`
- `?:`

As a member function with one parameter

```
bool operator<(const Quarterback& QB)
{
  std::cout << "Is " << this->qbName
            << " < " << QB.qbName << std::endl;

  if (qbAtt  < QB.qbAtt  &&
    qbComp < QB.qbComp &&
    qbYds  < QB.qbYds  &&
    qbTd   < QB.qbTd)
    return true;
  else
    return false;
}
```

As a non member function with two parameters

```
bool operator<(const Quarterback& QB1, const Quarterback& QB2)
{
  std::cout << "Is " << QB1.qbName
            << " < " << QB2.qbName << std::endl;

  if (QB1.qbAtt  < QB2.qbAtt  &&
      QB1.qbComp < QB2.qbComp &&
      QB1.qbYds  < QB2.qbYds  &&
      QB1.qbTd   < QB2.qbTd)
      return true;
  else
      return false;
}
```

What happens when I make this function a non member function?

It is asking to access access private member data…..

Operator Overloading

Overloading the relational operators

The test that determines if one object is less than or greater than another object is determined by the programmer. Those tests are written into the operator overload function.

Overloading the stream insertion/extraction operators

<< and >> can be overloaded to accept input or print output based on rules defined by the programmer. Those test are written into the operator overload function.

Exception Handling

```
vector<int> WholeNumbers={0,1,2,3,4};

for (int i = 0; i <= WholeNumbers.size(); i++)
{
   try
   {
      cout << WholeNumbers.at(i) << endl;
   }
   catch (out_of_range& ex)
   {
      cerr << "An exception occurred: " << ex.what() << endl;
   }
  }

  cout << "Even if an exception occurs, life goes on" << endl;
```

Introduction to Exception Handling

```
catch (out_of_range& ex)
{
    cerr << "An exception occurred: " << ex.what() << endl;
}
```

The catch block declares a type (`out_of_range`) and an exception parameter (`ex`) that it receives as a reference.

`ex` is the caught exception object

- catching an exception by reference increases performance by preventing the exception object from being copied when it is caught

Introduction to Exception Handling

```
catch (out_of_range& ex)
{
    cerr << "An exception occurred: " << ex.what() << endl;
}
```

`what()` is a member function of the exception object

`what()` will get the error message that is stored in the exception object and display it

Once the message is displayed, the exception is considered handled and the program continues with the next statement after the `catch` block's closing brace.

Inheritance

```
public class Shape
{
    public Shape(String name)
    {
        shapeName = name;
    }
```

```
class Shape
{
   public :
      Shape(std::string name="BaseShape") : ShapeName{name}
      {
      }
```

Inheritance

```
private String shapeName;
private double dim1;
private double dim2;
private String color;

public String xxxx;
public double yyyy;
```

```
private :
    std::string ShapeName;
    float dim1;
    float dim2;
public :
    std::string xxxx;
    float yyyy;
```

Inheritance

```
public class Circle extends Shape
{
    public Circle(String name, double radius)
    {
        super(name);
        setDims(radius, radius);
    }
```

```
class Circle : public Shape
{
   public:
      Circle(std::string name, float radius=0)
      : Shape(name)
      {
         set_dims(radius, radius);
      }
```

```
public CokeMachine(String name, int cost, int change, int inventory)
{
    machineName = name;
    CokePrice = cost;
    changeLevel = change;
    inventoryLevel = inventory;
}

public CokeMachine()
{
    machineName = "New Machine";
    CokePrice = 50;
    changeLevel = 500;
    inventoryLevel = 100;
}
```

```
CokeMachine::CokeMachine(std::string name, int cost, int change, int inventory)
            : machineName{name}, CokePrice{cost},
              changeLevel{change}, inventoryLevel{inventory}
            {}
public :
      CokeMachine(std::string="New Machine", int=50, int=500, int=100);
```

```
public String getMachineName()
{
    return machineName;
}

public void setMachineName(String newMachineName)
{
    machineName = newMachineName;
}
```

```
std::string CokeMachine::getMachineName(void)
{
      return machineName;
}

void CokeMachine::setMachineName(std::string newMachineName)
{
      machineName = newMachineName;
}
```

```
System.out.println(MyCokeMachine);

cout << MyCokeMachine;


std::ostream& operator<<(std::ostream& output, const CokeMachine& CM)
{
      output << "\n\nMachine Name : " << CM.machineName
              << "\n\nCurrent Inventory Level "
              << CM.inventoryLevel
              << "\nMax Inventory Capacity  "
              << CM.maxInventoryCapacity
              << "\nCurrent Change Level    "
              << CM.displayMoney(CM.changeLevel)
              << "\nMax Change Capacity     "
              << CM.displayMoney(CM.maxChangeCapacity)
              << "\nCurrent Coke price is   "
              << CM.displayMoney(CM.CokePrice);

      return output;
}
```

```
while (getline(HouseFH, HouseLine))
{
   if (rand() % 2)
   {
      Houses.push_back(new CandyHouse{FileLine, CRM});
   }
   else
   {
      Houses.push_back(new ToothbrushHouse{FileLine, CRM});
   }
}
```

```
while (FileReader.hasNextLine())
{
    FileLine = FileReader.nextLine();

    if (rn.nextInt(2) == 0)
        Houses.add(new CandyHouse(FileLine, CRM));
    else
        Houses.add(new ToothbrushHouse(FileLine, CRM));
}
```