CSCI 390 – Special Topics in C++

Lecture 7

9/11/18

Time To Turn Off Cell Phones

Functions (cont)
Overloading Functions

- In C++, the parameter types are part of the function name:
  - The following functions have different names because the type of their only parameters differ:
    - `void Func(uint32_t Parm);`
    - `void Func(float Parm);`
    - `Void Func(double Parm);`

Functions (cont) Overload Example

//main.cpp:
```
#include "Function.h"

int main(void)
{
  Func(1u);
  Func(1.5);

  return 0;
}
```

Console:
```
Thank you for the uint32_t with value: 1
Thank you for the double with value: 1.5


...Program finished with exit code 0
Press ENTER to exit console.
```

//Function.h:
```
#include <cstdint>

void Func(uint32_t Parm);
void Func(double Parm);
```

//Function.cpp:
```
#include <iostream>
#include "Function.h"

void Func(uint32_t Parm)
{
  std::cout << "Thank you for the uint32_t with value: " << Parm << std::endl;

  return;
}

void Func(double Parm)
{
  std::cout << "Thank you for the double with value: " << Parm << std::endl;

  return;
}
```

Functions (cont)
Returning A Value

//main.cpp:
```
#include <iostream>
#include "Helper.h"
#include "Function.h"

int main(void)
{
  std::cout << "Area of circle with radius " << 1.0 <<
    " is: " << AreaOfCircle(1.0) << std::endl;

  std::cout << "Area of circle with radius " << 2.0 <<
  " is: " << AreaOfCircle(2.0) << std::endl;

  return 0;
}
```

//Function.h:
```
double AreaOfCircle(double Radius);
```

Console:
```
Area of circle with radius 1 is: 3.14159
Area of circle with radius 2 is: 12.5664


...Program finished with exit code 0
Press ENTER to exit console.
```

//Function.cpp:
```
#include <iostream>
#include <cmath>
#include "Function.h"

static const double&& pi{acos(-1)};

double AreaOfCircle(double Radius)
{
  return pi * Radius * Radius;
}
```

Functions (cont) Template Functions

- In C++ you can parameterize types in a function.
- These are called “template” functions.
- Syntax:

  `template<typename` <identifier> [**=**<default type>]**>**

  Rest of function goes here, and <identifier> can be used as a type. There can be more than one template type.
- Entire function goes in interface file.

Functions (cont)
Template Functions

//main.cpp:

```
#include <iostream>
#include "Function.h"

int main(void)
{
  std::cout << "Area of circle with radius " << 1.0 <<
    " is: " << AreaOfCircle<>(1.0f) << std::endl;

  std::cout << "Area of circle with radius " << 2.0 <<
    " is: " << AreaOfCircle<>(2.0) << std::endl;

  std::cout << "Area of circle with radius " << 3.0 <<
    " is: " << AreaOfCircle<long double>(3.0) << std::endl;

  return 0;
}
```

//Function.cpp:

Console:

```
Thank you for the Type: float, Length: 4
Area of circle with radius 1 is: 3.14159
Thank you for the Type: double, Length: 8
Area of circle with radius 2 is: 12.5664
Thank you for the Type: long double, Length: 16
Area of circle with radius 3 is: 28.2743


...Program finished with exit code 0
Press ENTER to exit console.
```

//Function.h:

```
#include <iostream>
#include "Helper.h"

template<typename T>
T AreaOfCircle(T Radius);

template<typename T>
T AreaOfCircle(T Radius)
{
  std::cout << "Thank you for the " << DUMPTYPE(Radius) << std::endl;

  return T(3.14159265358979323846264433) * Radius * Radius;
}
```

Template functions are defined in the interface file!

Functions (cont) The **`main`** Function

- **`main`** has 2 and often 3 overloaded prototypes:
  - **`int main(void);`**
  - **`int main(int`** argc**`, char *`**argv**`[ ]);`**
  - And, often:
    `int main(int` argc**`, char *`**argv**`[ ], char *env[ ]);`**
- argc is the number of command line args.
- argv is the command line args.
  - argv[0] is the executable name.
- env is the environment variables.
  - Ends with **`nullptr`**.

Functions (cont)
`main` Example

//main.cpp:

```
#include <iostream>

extern char**environ;

int main(int argc, char *argv[])
{
  std::cout << "There are " << argc << " arg(s)." << std::endl;
  std::cout << "The executable is " << argv[0] << std::endl;
  std::cout << "The first environment variable is " << environ[0] <<
    std::endl;

  return 0;
}
```

Console:

```
There are 1 arg(s).
The executable is /home/a.out
The first environment variable is SUDO_GID=0


...Program finished with exit code 0
Press ENTER to exit console.
```

Operators

- Operators are the basic building block for sequential program flow.
- They perform a *function* on operands:
  - 1 operand (unary operator)
  - 2 operands (binary operator)
  - 3 operands (ternary operator)

Operators (cont)

- Operators have:
  - Precedence (Priority)
  - Associativity (order of evaluation)
    - Either left to right, or right to left
  - Perform a function on operands.
  - Return a value of some type
  - Optionally have side effects.

Operators (cont) Summary

Operator precedence and associativity

Precedence		Assoc
1	`::`	LR
2	`( )` `[ ]` `->` `.` `x++` `x--`	LR
3	`!` `~` `+<unary>` `-<unary>` `++x` `--x` `*<dereference>` `&<address of>` `sizeof` `(type)<cast>` `new` `delete`	RL
4	`.*` `->*`	LR
5	`*` `/` `%`	LR
6	`+` `-`	LR
7	`<<` `>>`	LR
8	`<` `<=` `>` `>=`	LR
9	`==` `!=`	LR
10	`&`	LR
11	`^`	LR
12	`|`	LR
13	`&&`	LR
14	`||`	LR
15	`?` `:`	RL
16	`=` `+=` `-=` `*=` `/=` `%=` `>>=` `<<=` `&=` `^=` `|=`	RL
17	`,`	LR

Operators (cont)
Priority 1, Left to Right

- `::` (qualification)
  - Unary `::`<operand>
    - Accesses <operand> in global scope.
  - Binary <scope>::<operand>
    - Accesses <operand> in <scope>.
- Returns <lvalue>

Operators (cont) Priority 2, Left to Right

- `(`<expression>`)` (Overide priority)
  - Returns: <expression>
  - Side Effects: None
- <identifier>`[` <expression>`]` (Index operator)
  - Returns: `*(`<identifier> `+` <expression>`)`
  - Side Effects: None
  - This is an <lvalue>

Operators (cont) Priority 2, Left to Right

- <lvalue>**++** (Post-increment)
  - Returns: Value at <lvalue> (This is an <rvalue>.)
  - Side Effects: Increments <lvalue>
- <lvalue>**−−** (Post-decrement)
  - Returns: Value at <lvalue> (This is an <rvalue>.)
  - Side Effects: Decrements <lvalue>

Operators (cont)
Priority 3, Right To Left

- **!**<rvalue> (Logical not)
  - Returns: 1 if <rvalue> == 0, 0 otherwise
  - Side Effects: None
- **~**<rvalue> (Bitwise not)
  - Returns: <rvalue> with bits flipped
  - Side Effects: None