UNIVERSITY of LOUISVILLE
dare to be great

CPP Chapters 2-4:
Data Types, Control and Functions

CECS130
Introduction to Programming Languages
Dr. Roman V. Yampolskiy

Outline

- Chapter 2 – boolean data type
- Chapter 3 – n/a
- Chapter 4 –
  - Function Overloading
  - Default Arguments
  - Scope Resolution
  - Static Variables

`bool` Data Type

- `bool` type
  - Has two values, `true (1)` and `false (0)`
  - Manipulate logical (Boolean) expressions
- `true` and `false` are called logical values
- `bool`, `true`, and `false` are reserved words

Bool: Example

```
#include <iostream>
using namespace std;
int main(void) {
    bool status = false;
    if (status) {
       cout<<"Status is True "<<status<<endl;
    } else {
       cout<<"Status is False "<<status<<endl;
    }
    system("pause");
    return 0;
}
```

Bool: Example 2

```
#include <iostream>
using namespace std;
int main() {
  bool b;
  b = false;
  cout <<  "b is " << b << endl;
  b = true;
  cout <<  "b is " << b << endl;
  if(b)                               // control the if statement
     cout <<  "This is executed.\n";
  b = false;
  if(b)
     cout <<  "This is not executed.\n";
  cout <<  "10 > 9 is " << (10 > 9) << endl;
// outcome of a relational operator is a true/false value
  system("pause");
  return 0;
}
```

Output:

```
b is 0
b is 1
This is executed.
10 > 9 is 1
```

Chapter 3: Control Statements

- No new material
  - Everything in chapter 3 of your CPP book was already covered in your C book

Function Overloading

- C++ supports writing more than one function with the same name but different argument lists. This could include:
  - different data types
  - different number of arguments
- The advantage is that the same apparent function can be called to perform similar but different tasks

Function Overloading: Prototypes

```
void swap (int *a, int *b) ;
void swap (float *c, float *d) ;
void swap (char *p, char *q) ;
```

Function Overloading: Definitions

```
void swap (int *a, int *b)  {
    int temp;
    temp = *a;
    *a = *b;
    *b = temp;
}
void swap (float *c, float *d) {
    float temp;  temp = *c;  *c = *d;  *d = temp;
}
void swap (char *p, char *q) {
    char temp;  temp = *p;  *p = *q;  *q = temp;
}
```

Function Overloading: Calls

```
int main ( ) {
    int a = 4, b = 6 ;
    float c = 16.7, d = -7.89 ;
    char p = 'M' , q = 'n' ;
    swap (&a, &b) ;
    swap (&c, &d) ;
    swap (&p, &q) ;
 }
```

NUMBER OF PARAMETERS EXAMPLE

```
#include<iostream>

 //FUNTION PROTOTYPES
 int func(int i);
 int func(int i, int j);

 void main(void) {
     cout<<func(10);                    //func(int i)is called
     cout<<func(10,10);                 //func(int i, int j) is called
 }

 int func(int i) {
     return i;
 }

 int func(int i, int j)   {
     return i+j;
 }
```

TYPE OF PARAMETERS EXAMPLE

```
#include<iostream>

 //FUNTION PROTOTYPES
 int func(int i);
 double func(double i);

 void main(void)   {
      cout<<func(10);                    //func(int i)is called
      cout<<func(10.201);                //func(double i) is called
 }

 int func(int i)   {
      return i;
 }

 double func(double i)   {
      return i;
 }
```

Functions: Default Arguments

- In C++ you can specify a default value for some parameters to the function.
- When an argument is omitted in a function call, the default value of that argument is automatically passed in the call.
- Default arguments must be the rightmost (trailing) arguments.
- Default arguments can be preceded by non-default arguments

Default Arguments: An Example

```
// Using default arguments
#include <iostream>

// Calculate the volume of a box
int boxVolume(int length = 1, int width = 1,
              int height = 1) {

  return (length * width * height);

}
```

Default Arguments: An Example of Calls

```
main() {
   cout << "The default box volume is: "
        << boxVolume()
        << "\n\nThe volume of a box with length 10,\n"
        << "width 1 and height 1 is: "
        << boxVolume(10)
        << "\n\nThe volume of a box with length 10,\n"
        << "width 5 and height 1 is: "
        << boxVolume(10, 5)
        << "\n\nThe volume of a box with length 10,\n"
        << "width 5 and height 2 is: "
        << boxVolume(10, 5, 2)
        << '\n';
   return 0;
}
```

Output

```
The default box volume is: 1

The volume of a box with length 10,
width 1 and height 1 is: 10

The volume of a box with length 10,
width 5 and height 1 is: 50

The volume of a box with length 10,
width 5 and height 2 is: 100
```

Default Arguments: Example

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

void f(char *s1, char *s2, int len = 0);

int main() {
  char str1[80] = "This is a test";
  char str2[80] = "0123456789";

  f(str1, str2, 5);
  f(str1, str2);

  system("pause");
  return 0;
}

void f(char *s1, char *s2, int len){
  cout << s1;
  cout << " " << len << " ";
  cout << s2<<endl;
}
```

Output:

```
This is a test 5 0123456789
This is a test 0 0123456789
```

Default Arguments: **Ambiguous function call**

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

int myfunc(int i);
int myfunc(int i, int j=1);

int main() {
  cout << myfunc(4, 5) << " "; // unambiguous
  //cout << myfunc(10); // ambiguous- will not work

  system("pause");
  return 0;
}

int myfunc(int i) {
  return i;
}

int myfunc(int i, int j){
  return i*j;
}
```

Scope Resolution Operator

- The :: (scope resolution) operator is used to qualify hidden names so that you can still use them
- You can use the unary scope operator if a global scope name is hidden by an explicit declaration of the same name in a block
- Basically ::can be used to overcome some scope limitations

Scope Resolution Operator: Example

```
int count = 0;              // global variable

int main(void) {

    int count = 0;          // local variable
    ::count = 1;            // set global count to 1
    count = 2;              // set local count to 2
    return 0;
}
```

Static and Automatic Variables

- Automatic variable - memory is allocated at block entry and de-allocated at block exit
- Static variable - memory remains allocated as long as the program executes
- By default, variables declared within a block are automatic variables
- Declare a static variable within a block by using the reserved word **static**

Static Variables

- The syntax for declaring a static variable is:

```
static dataType identifier;
```

- The statement

```
static int x;
```

declares x to be a static variable of the type int

- Static variables declared within a block are local to the block
- Their scope is the same as any other local identifier of that block
- Static variables have lifetimes which last until the end of the program making it possible to create functions with memory

Static Variables: Example

```
int counter() {
    static int x = 0;
    x++;
    return x;
}
```

- x is initialized only once regardless of how many times counter() is called
- counter() knows (remembers) how many times it was executed.

The End!