C++ Foundations

Eden Burton <ronald.burton@senecacollege.ca>

17/09/2015

A Few More Notes on Types

Boolean Type - new to C++

- keywords true and false
- int to bool conversions: true = 1, false = 0
- treats integer of 0 as false , all non-zero integers as true

Compound Types

- composed of other types
- `struct` or **`class`**

Auto Type

- allows to be determined base on type of initializing expression

Boolean Example

Declarations vs Definitions

- declaration , associates an entity with a type
- definition, declaration that binds meaning to an identifier
  - definitions can only occur once per block

```
struct Transaction; // declaration

// definition - binds Transaction to a struct
struct Transaction {
   int acct;
   char type;
   double amount;
}
```

Declaration Example

Scope

- portion of program where identifier is visible
  - global visible to entire program
  - block visible only within a code block

```
int someFunction() {
    ...
    for (i = 0; i < 8; i++) {
        int j = 0;
        ...  // some code
        j = 18; // ok:  j is in scope
    }
    j = 12;  // error:  j is out of scope
}
```

Shadowing

- scopes can be nested
- inner scope declarations take priority over outer ones
  - global visible to entire program
  - block visible only within a code block

```
int someFunction() {
    int j = 5
    for (i = 0; i < 8; i++) {
        int j = 0;
        ...  // some code
        j = 18; // "inner j" is used
    }
    j = 12;  // "outer j" is used
}
```

Shadowing Example

Functions

Function Prototypes are required

- function's return type, its identifier and all of its parameter types.
- classification

```
// what does printf mean
int main() {
    printf("Hello C++\n");
}
```

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

int main() {
    printf("Hello C++\n");
}
```

Function Overloading

A function identifiers can have multiple meanings

- are distinguished by the number and type of arguments

```
void display(int x) {
     cout << x << endl;
 }

void display(int* x, int n) {
     for (int i = 0; i < n; i++)
          cout << x[i] << ' ';
     cout << endl;
 }
```

Function Overloading Example

Defining Default Arguments For Functions

type identifier(type[, ...], type = value)

- require that default parameters are the right-most ones

```
void display(int a, int b = 5, int c = 0);
```

Passing Arguments To Functions

type identifier(type[, ...], type = value)

- pass-by-value, argument is a copy of the variable
- pass-by-address, argument is a pointer to variable
- pass-by-reference, argument is an alias of the variable

```
// pass-by-value
void swap ( char a, char b );

// pass-by-address
void swap ( char *a, char *b );

// pass-by-reference
void swap ( char &a, char &b );
```

Member Functions

"...recall that a structure (or class) is composed of data and member functions used to modify it..."

```
class Box {
   double length;
   double breadth;
   double height;
   double volume;

   double getVolume();
   double setHeight(double h);

 };
```

More on Member Functions

Member Function Classifications

- accessor methods, answer question about object state without modifying it
- mutator methods, they modify object state
- special, create, assign and destroy objects

```
// declaration usually put in the header file
class Box {
   double length;
   double breadth;
   double height;
   double volume;

   double getVolume() const;     // accessor
   double setHeight(double h);   // mutator
};
```

Privacy

Accessibility Labels

- private: - prevents external access by clients
- public: - allows client access

```
struct Student {
      int no;
      char grade[14];
      void display() const;
 };

 class Student {
      int no;
      char grade[14];
      void display() const;
 };
```

- `struct` makes members public by default
- **`class`** makes members private by default

More Privacy

– labels set viability until another label changes it

```
struct Student {
  private:
    int no;
    char grade[14];
  public:
    void display() const;
};
...
int main() {
   Student st;
   st.display()    // ok
   st.no;        //error, cannot access
}
```