Lecture 8 - Classes and Objects

Default Member Initializers

Many times we need simple initialization for class data members

- Prior to C++11, initialization was only allowed through constructors
- This is defined starting with the 2011 C++ standard (C++11)

DayOfYear.h

```
class DayOfYear{
  private:
    int day = 1;
    int month = 1;
  public:
    void setDay(int d);
    void setMonth(int m);
    void print();
};
```

Compiling code using **default member initialization** requires telling the compiler to use the C++11 standard

> g++ -std=c++11 main.cc

Destructors

Constructors are called when objects are created

- C++ also automatically calls a function when an object is destroyed/deleted

Destructors are functions that you write and are automatically called when objects are about to disappear

- The **destructor** does not destroy the object or clear the memory
  - The **destructor** simply defines the "final" actions of the destructor
- **Destructors *must*** have the same name of the class, ***and*** have a tilda (~) in front of the name
  - Destructors are members of the class
  - Destructors have **no** return type

    - Destructors are usually public
- **Destructors** have **no** parameters
    - A way to think about this is that there is only the *default constructor*
- There can only be **one** destructor
- If there is no **Destructors** defined, C++ will define an empty (default) one for you

DayOfYear.h

```
class DayOfYear{
  private:
    int day = 1;
    int month = 1;
  public:
    DayOfYear();  //constructor
    ~DayOfYear(); //destructor
    void setDay(int d);
    void setMonth(int m);
    void print();
};
```

DayOfYear.cpp

```
DayOfYear::~DayOfYear(){

}
```

Notes:

- The provided constructor does nothing

'This' Pointer

Every method is given the address of the object on which it is invoked

- The *address* is stored in the **this** pointer
    - Initialized to point to the object on which the method is invoked
- Think of it as (read-only) variable of type pointer to the class
    - `DayOfYear* this ;`
- No way to initialize **this**
    - Compiler defines the **this** pointer for you
    - No way to redefine/modify **this** pointer
- The **this** pointer is always defined
    - Will never be `null` if object is defined
- Accessible only within member functions

DayOfYear.cpp

```
void DayOfYear::setDay(int d){
  day = d;
}
void DayOfYear::setDay(int d){
  day = d;
  (*this).day = d;
  this->day = d;
}
```

Notes:

1. The three lines of code in `DayOfYear::setDay` above have the *same* functionality
2. You need brackets for `(*this).day`
    - Field access operator `.` has higher precedence than dereference operator `*`
3. Arrow operator `->` similar idea to arrow operator in C
    - Access value of field at address of object
4. `this.day` is not valid
    - Syntax error, attempting to access field of address

Notes about printing objects:

- Can print address `this`
- No way to print `(*this)`
    - For example, `cout << *this << endl;`
    - `*this` refers to the value of the object, which doesn't have an identifiable print value (for the compiler)
        - Should printing this print the data members, or an arbitrary string?
        - Not defined
- One way to print objects
    - Overload insertion operator `<<`
    - No `toString()` like in Java

Type Conversion

Given the class definition and main.cpp:

DayOfYear.h

```
class DayOfYear{
  private:
    int day = 1;
    int month = 1;
```

```
  public:
    DayOfYear();
    DayOfYear(int d,int m);
    ~DayOfYear();
    void setDay(int d);
    void setMonth(int m);
    void print();
};
```

main.cpp

```
DayOfYear x("6","8")
```

Will result in compile-time error.

However,

main.cpp:

```
DayOfYear x(5.1,2.34)
```

Will compile successfully.

Why?

- C++ compiler attempts to match types to function parameters
  - In this case, `float` types can be converted easily to `int` types by flooring.
  - Whereas the **type conversion** from `string` to `int` is not defined

Address of Object

Outside of member functions, you can access address by using the referencing operator `&`

- Get address of `object` with `&object`
- Same as C for referencing pointers

Garbage Collection

C++ objects defined on the memory **stack**

- LIFO (Last-in, First-out)
- The first member functions to be deleted are the last member functions to be created
  - Deletion is *reverse* order of creation