17 - Lecture - Introduction to C++

C++ textbook

C++ Primer 5th Ed., by Lippman, et al.

See the following link for recommendations for other more advanced-level C++ books.

http://stackoverflow.com/questions/388242/the-definitive-c-book-guide-and-list

Reading assignment

C++ Primer 5th Ed., by Lippman, et al. (we’ll call it Lippman5):

- chapter 7: classes
- chapter 13.1.1 - 13.1.3: copy and assignment
- chapter 14.1 - 14.6: operator overloading

Here are the corresponding sections in C++ Primer 4th Ed. (Lippman4):

- chapter 12: classes
- chapter 13.1 - 13.3: copy and assignment
- chapter 14.1 - 14.5, 14.7: operator overloading

Why C++?

Features:

object-oriented programming facilities
- user-defined types (classes)
- polymorphism (inheritance)

generic programming (templates)

exceptions

full-blown standard library
- containers
- algorithms

They’re all nice, but the real reason for many people:

> "I’m so sick and tired of char*. Can somebody give me a STRING?"

Our approach

We can’t possibly cover all C++ in a few weeks, so we’ll concentrate on:

how the fundamental language facilities work

- cornerstone for writing correct and safe C++ code

standard library essentials

- extremely useful for getting things done
- we’ll need to cover templates as a prerequisite

practical example-based approach

- class design as well as C++ syntax

Unfortunately we’ll have to skip these:

- polymorphism

- exceptions

- and other gzillion nifty features of C++

String in C

1) A string allocated on the stack:

```
char buf[100];
strcpy(buf, "hello ");
strcat(buf, "world");
...
```

2) A string allocated on the heap:

```
char *buf = (char *)malloc(100);
strcpy(buf, "hello ");
strcat(buf, "world");
...
free(buf);
```

3) Using struct in C:

```
typedef struct {
        char *s;
        int len;
} String;

String *allocString(const char *s);
void deallocString(String *str);
int appendString(String *str, const char *s);

...

// this is how you use it:

String *p = allocString("hello");
if (!appendString(p, "world"))
        die();
printf("%s", p->s);
deallocString(p);
```

4) It would be nice to have something like this:

```
        String s = "hello";
        s = s + "world";

        // and not worry about deallocating the string
```

Review of some important concepts

Declaration v. Definition

Declaration tells the comiler the name & type of an object, which is defined somewhere else:

```
        extern int x; // refers to x in another file

        int f(int x);

        struct MyList;

        class MyString;

        template<class T>
        class MyTypedList;
```

Definition:

```
        int x; // memory is allocated here

        functions with code body

        structs & classes with members listed
```

Stack v. Heap allocations

```
        struct Pt {
                int x;
                int y;
        };
```

In C:

```
        // stack allocation
        struct Pt p1;
        // p1 goes away at the end of its scope

        // heap allocation
        struct Pt *p2 = malloc(sizeof(struct Pt));
        ...
        free(p2);
```

In C++:

```
        // stack allocation
        struct Pt p3(0,0);
        // p3 gets destructed at the end of its scope

        // heap allocation
        struct Pt *p4 = new Pt(0,0);
        ...
        delete p4;
```

Pass-by-value v. Pass-by-ref

```
  1) f(struct Pt p)

  2) f(struct Pt *p)

  3) f(struct Pt &p)
```

C++ constructs: new & delete operators and references

```
        // stack-allocated objects
        String s1;
        String s2 = String();
        String s3("hello");

        // heap-allocated objects
        String *p1 = new String();

        // heap-allocated array of objects
        String *a1 = new String[10];

        // pointer
        String *p2 = p1;
        String *p3 = &s3;

        // reference
        String& r3 = s3;
        String& r1 = *p1;

        // more stack-allocated objects,
        // which are duplicates of the existing objects
        String s4(r3);
        String s5 = s3;

        // heap-allocated object must be deleted
        delete p1;

        // heap-allocated array of objects must be deleted differently
        delete [] a1;
```

C++ Basic 4: ctor, dtor, copy, op=()

  - getting these right is the half the battle

    1) Constructor

- Decide the arguments (Provide default constructor in most cases)
- Cover all possible argument values
- Properly initialize all data members and base classes

2) Destructor

- Properly deallocate all data members

3) Copy constructor

- Called in three cases

4) Operator=()

- Called in assignment expressions

- Compiler generates them when you don’t provide them
  - may not be what you want
  - declare them private if you don’t want them

String class example

```
class MyString {

    public:

        // default constructor
        MyString();

        // constructor
        MyString(const char* p);

        // destructor
        ˜MyString();

        // copy constructor
        MyString(const MyString& s);

        // assignment operator
        MyString& operator=(const MyString& s);

        // returns the length of the string
        int length() const { return len; }

        // operator+
        friend MyString operator+(const MyString& s1, const MyString& s2);

        // put-to operator
        friend ostream& operator<<(ostream& os, const MyString& s);

        // get-from operator
        friend istream& operator>>(istream& is, MyString& s);

        // operator[]
        char& operator[](int i);
        // operator[] const
        const char& operator[](int i) const;

    private:

        char* data;

        int len;
};
```