CPSC 427: Object-Oriented Programming

Michael J. Fischer

Lecture 12
February 25, 2016

Pointer Arithmetic

Bells and Whistles

Classes

Pointer Arithmetic

Meaning

Addition and subtraction of a pointer and an integer gives a new pointer.

```
int a[10];
int* p;
int* q;
p = &a[3];
q = &a[5];
// q-p == 2
// p+1 == &a[4];
// q-5 == &a[0];
// What is q-6?
```

Implementation

Pointers are represented internally by memory addresses.

The meaning of `p+k` is to add `k*sizeof *p` to the address stored in `p`.

Example: Suppose `p` points to a `double` stored at memory location 500, and suppose `sizeof(double) == 8`. Then `p+1` is a pointer to memory location 508.

508 is the memory location of the first byte following the 8 bytes reserved for the double at location 500.

If `p` points to an element of an *array* of `double`, then `p+1` points to the *next* element of that array.

Bells and Whistles

Optional parameters

The same name can be used to name several different member functions if the *signatures* (types and/or number of parameters) are diffent. This is called overloading.

Optional parameters are a shorthand way to declare overloading.

Example

```
int myfun( double x, int n=1 ) { ...  }
```

This in effect declares and defines two methods:

```
int myfun( double x ) {int n=1; ...}
int myfun( double x, int n ) {...}
```

The body of the definition of both is the same.
If called with one argument, the second parameter is set to 1.

const

const declares a variable (L-value) to be readonly.

```
const int x;
int y;
const int* p;
int* q;

p = &x;  // okay
p = &y;  // okay
q = &x;  // not okay -- discards const
q = &y;  // okay
```

const implicit argument

`const` should be used for member functions that do not change data members.

```
class MyPack {
private:
  int count;
public:
  // a get function
  int getCount() const { return count; }
...
};
```

Operator extensions

Operators are shorthand for functions.

Example: `<=` refers to the function `operator <=()`.

Operators can be overloaded just like functions.

```
class MyObj {
  int count;
  ...
  bool operator <=( MyObj& other ) const {
      return count <= other.count; }
};
```

Now can write `if (a <= b) ...` where `a` and `b` are of type `MyObj`.

Classes

What is a class?

- A collection of things that belong together.
- A struct with associated functions.
- A way to encapsulate behavior: public interface, private implementation.
- A way to protect data integrity, providing world with functions that provide a read-only view of the data.
- A data type from which objects (instances) can be formed. We say the instances belong to the class.
- A way to organize and automate allocation, initialization, and deallocation of storage.
- A way to break a complex problem into manageable, semi-independent pieces, each with a defined interface.
- A reusable module.