Advanced Computer Contest Preparation

Lecture 25

# REVIEW: POINTERS AND OBJECTS

# Pointers

- A pointer is a variable that stores a memory address
- In the form of `type * ptr`
  - `type` can be anything, including another pointer!
    - If `type` was `int*`, then a pointer to `type` would be `int**`

# Pointer Operators

- Dereference Operator (*)
  - Gets the variable that a pointer is pointing to
- Address-of Operator (&)
  - Gets the memory address of the variable

# References

- A reference allows for more than one identifier (name of variable) to be associated with the same memory address
- In the form of `type & ref`
- Example:
  - `int a = 5;`
  - `int &b = a;`
  - If the value of either `a` or `b` changes, the other will change as well

# Pointers and Arrays

- When arrays are declared, a consecutive section of memory is allocated to it
- An array's identifier is actually a pointer to the first element
- By adding an offset value to this pointer, and dereferencing, we can get the elements we want
- Note: `ar[i]` is the same as `*(ar+i)`, meaning that `i[ar]` is valid

# Function Prototypes

- Normally, C++ functions cannot access functions that are declared below them
- To alleviate this problem, we use function prototypes
- Before implementation of any function, state function signatures at the top
  - e.g. `void func(int param);`
- Functions can now call other functions implemented afterward

# Function Parameters

- Pass by value
  - By default, when arguments are sent into a function, they are copied
  - Any changes to the copies are not reflected when the function exits
- Pass by reference
  - By using `&` after a parameter type, we pass by reference instead
  - Arguments are not copied, any modifications will affect original variables
- Example: `void func(int a, int & b)`
  - `a` is passed by value, `b` is passed by reference
- Tip: Pass pointer by reference is `type *&param`

# const

- `const` keyword makes something unchangeable, or read-only
- Simple example: `const int a = 4;`
- `const` in pointers:
  - `const type * ptr`
    - non-const pointer to const type
    - `type const * ptr` is also equivalent
    - Pointer can point to different variables, but cannot modify said variables
  - `type * const ptr`
    - const pointer to non-const type
    - Pointer can modify pointed variable but cannot point to another variable
  - `const type * const ptr`
    - const pointer to const type

# Classes

- OOP in C++ works similar to Java
- Two main types of classes:
  - **`class`** – members are by default private
  - **`struct`** – members are by default public
- Inheritance is in the form

```
class myClass: public Parent
```

- C++ supports multiple inheritance

```
class myClass: public Parent1, public Parent2
```

# Class Access Modifiers

- Refers to visibility of members of a class
  - i.e. **`public, protected, private`**
- In the form of:

```
access_specifier:
  members
```

- Example:

```
public:
    int a;
    double b;
private:
    char c;
```

# Constructors

- Constructor style similar to Java
- In the form of `Classname(args){}`
- C++ allows initializer lists
  - Allows for quick setting of object variables
  - In the form of
    `Classname(args):var1(param1),var2(param2),...{}`
  - Example:

```
struct Point{int x,y;
Point(int _x, int _y):x(_x),y(_y){}};
```

# Objects as Variables

- C++ allows you to have direct access to objects
- Note: in Java, the only way to access objects is via pointers; you do not have direct access to objects
- Declaration form: `type obj(params);`
- No round brackets required for a call to the default constructor (no parameters)
  - `type obj;`
- Note the lack of the `new` keyword
- Objects can never be null

# Pointers to Objects

- Pointers to objects function in a similar way to how objects work in Java
- To access members of the object via pointer, use the arrow operator **`ptr->member`**
  - Syntactic sugar for **`*(ptr).member`**
- To instantiate a new object via a pointer, use the **`new`** keyword
  - **`Object *obj = new Object();`**

# `this`

- **`this`** in C++ is a *pointer* to the object itself
- Because **`this`** is a pointer, we operate on it as a pointer, not as an actual object

# Operator Overloading

- Operators (such as `+`,`-`,`<`,`=`,`==`) can be redefined in classes
- In the form of
  - `returntype operator[your operator](params) const`
  - e.g. `bool operator==(const Object &o) const`
  - Code might not compile if `const` at the end is forgotten (makes the function const)
- A nice way to be able to sort objects, push into priority queues, etc.

# Memory Management

- When declare an object (not pointer), it is automatically deleted when you leave its scope
- However, when you declare a pointer to an object and call a constructor, it does not get deleted when you leave the pointer's scope
- If these objects aren't managed properly, you may encounter memory leaks
- Use the **`delete`** keyword on a pointer to free the memory it is pointing to
  - e.g. **`delete ptr`**

# Practice

- Implement a *binary search tree*
- Each node of the tree should have value, right child, and left child (optionally parent)
- BST property: all values on the left are < than the current node's value, all values on the right are >= than the current node's value
- Implement two operations:
  - Insert node with given value into tree
  - Find node in tree with value
- Bonus:
  - Get the $x^{th}$ largest element in the tree
  - Get the index of a given value in the tree

# THANK YOU!