C++ Basics

Structs and Class

In vanilla C, structs are collections of variables, originally created for custom typing. In C++, structs and classes are the same and have the same function as any other object oriented language. The only difference between the two is that in structs, variables and methods are by default public and in classes they are by default private. This basically means that for the purpose of competitive programming we almost always want to use structs.
The following is an example of a struct used to keep tract of a product on sale:

```
// this is a declaration of a struct
// it follows the general pattern of
// struct <struct_name> {
//     <member_type_1> <member_name_1>;
//     <member_type_2> <member_name_2>;
//     . . . . . .
//     <method_type_1> <method_name_1>(<params>) {<implementation>};
// };
struct product {
    string name;
    double price;
    bool avaliable;

    // since structs and classes are the same, we can define constructors
    product(string a_name, double a_price) {
        name = a_name;
        price = a_price;
        avaliable = true;
    };

    // we can also define our own comparison operators, default behavior
    // is very glitchy so it is recommended if you want to sort an array
    // of this type to define your own comparison
    bool operator < (const product &B) const {
        return price < B.price;
    };
};

// to create an element of the type product you can use a constructor
product apple("apple", 6.5);
// or in c++11 you can initialize any struct with list initialization
// by default the order of arguments provided is the order of struct
// members in the definition, but you can also specify
// omitted values are given the default value of the type
product banana = {"banana", 3.33, true}; // <- for our purposes use this
product orange = {.price = 3.33, .name = "orange", .avaliable = true};
```

Lambda Expression

Lambda expressions are inline anonymous functions especially helpful when using certain functions in `algorithms` when doing special comparison. The following example is an im-

plementation of something similar to filter in cpp:

```
#define T 5

void filterAbove(vector<int> &v) {
    // the syntax for a lambda expression is
    // [<capture group>](<argument list>) -> <ret type> { <code> }
    // the return type along with -> are often omitted for simple code
    // because the compiler can guess the return type.
    // the capture group is what the lambda function ``captures'' from the
    // surrounding namespace, for competitive programming purposes leave
    // it empty and just use #define statements up top.
    transform(v.begin(), v.end(), [](double d) { return d > T ? d : 0 });
}
```

Pointers

Don't use them on purpose... Just know that `*p` dereferences a pointer and `p+1` accesses the address at `p+` the size of the type of pointer that `p` is.