CSE 232
First Midterm, Overview:

1. Getting Started

1. we got started

2. First Program Stuff

1. Compiler vs. Intepreter
    a. do you know all the steps to create an executable?
2. Variables are declared
    a. indicate the type of the variable
    b. much of C++ (functions, STL) depend on types being available at compile time
        i. compile-time vs. run-time?
3. Symbols are overloaded to
    a. required context to figure it out
4. Basic program stuff
    a. what is an include file?
    b. what is a namespace
        i. does `using namespace std` do that I don't want?
        ii. how do I want you to do it?
        iii. what operator is `::`
    c. two types of comments
    d. what is a block? How is it indicated?
    e. what purpose does whitespace/indentation serve in C++
    f. `cout` and `cin`, insertion op and extraction op.
        i. what does `endl` do (two things)
5. what does it mean to have a function overloaded?

3. Types

We spend an awful lot of time on types

1. Do you know the basic, inbuilt types? What are their differences? Can they vary? What should you use when in doubt?
2. Ways to do initialization
    a. don't do any (what do you get?)
    b. assign, paren, block
        i. what are the differences?

<u>Type modifiers. Things we can add on top of the type</u>

3. `unsigned`, for what kind of types?
4. What does the compiler track with respect to variables?
5. References
    a. how indicated?
    b. what does a reference mean (w.r.t. the stuff the compiler tracks)
    c. is it a new object? does it have to be initialized (why or why not)?
    d. what types are `var1` and `var2`: `int & var1, var2;`
    e. what is the size of a reference type?

6. Pointers
    a. how indicated?
    b. what value does a pointer hold?
    c. is it a new object? Does it have to be initialized (why or why not)?
    d. what is the size of a pointer type (what question do you have to ask first)?
7. `&` and `*` in an expression
    a. what do these represent? Can you use them?
    b. what does the following print: `int *ptr; cout << *ptr;`
    c. what does the following print:
        i. `int x=10; int &r_x=x; int *p_x=&r_x; *p_x=5; cout<< x;`
8. Constants
    a. how indicated?
    b. what does it do to a value?
        i. can I assign it?
        ii. can I copy it?
    c. does it need to be initialized?
    d. can I remove it from a value?
    e. can I add it to a variable making reference to a non-constant value
        i. example???
    f. What two things can be constant in a pointer?
9. C++11 stuff
    a. what is a `typedef`?
    b. what does `auto` mean
        i. what are the rules here?

4. Expressions

1. `cout` formatting
    a. know the various ways to set things
        i. what is special about `setw`?
2. `cin` formatting
    a. what does `cin` take as a default separator?
    b. what does `noskipws` do?
3. Numeric ops (you basically know these)
    a. know what `int` on `int` division yields
    b. hex and oct, how indicated?
4. What is the way to do a cast in C++? Can you write one?
5. Assignment, it returns a value. What problems does that cause? Can you chain assignment? Why or why not?
6. Pre vs. Post increment/decrement.
    a. how indicated
    b. What difference does it make? Be very specific?
7. Compound Assignments, a shortcut: Any operator can precede an equal sign. It has the following meaning:
    a. a op= b means a = a op b
    b. a += b means a = a + b

c. a /= b means a = a/b

8. Booleans and Conditionals
    a. Truth and Falsity in C++.
        i. False is represented by 'empty' things in each type: 0 (int), 0.0 (float), ' ""(string)
        ii. If it isn't false (see above), then it represents true
9. Relational Operators, know what they are!
    a. they return true or false
10. Chained/compound comparisons **do not work like you want** in C++:
    a. is the following legal? `0 <= 15 <= 10;`
    b. if so what does it return?
11. Know the Logical Operators (`&&`, `||`, `!`)
12. Short circuiting. Logical operators do not return just true and false. They return the first value in an expression which makes the value of the expression clear. This is called short circuiting
    a. `1 || 2` returns 1
    b. `0 && 27` returns 0

5. Control

1. Selection
    a. do one statement (how to change)
    b. how do you know what `else` goes with what `if`

```
if (Boolean)
        statement;

    if (Boolean)
       statement;
    else
       statement;

    if(Boolean)
       statement;
    else if(Boolean)
       statement;
    else
```

2. Repetition
    a. While loop (top tested)
    b. `for` loop
        i. do you know the three parts?
        ii. how are they separated?
        iii. which are optional?
        iv. can you write an equivalent `while` for a `for` loop?
        v. what is the scope of a variable declared in a `for` loop?
3. what is the `switch` statement good for
    a. under what conditions would you use it?
    b. with what kind of data?

4. ternary operator
    a. can you write one
    b. what's special about it (compared to an `if` statement)

6. Functions

1. Functions are an encapsulation of a program. They are useful because they support:
    a. reusable code (can be used in many places)
    b. encapsulated code (details of implementation are hidden)
    c. portable code as modules/libraries (can be imported)
2. More on why functions?
    a. break a larger program down into smaller, understandable parts.
    b. for easier update or "refactoring". Refactoring, as applied to functions, takes a larger piece of code and breaks it down into smaller function pieces. This makes it easier to maintain.
3. How to write a function
    a. should do "one" thing. It represents one "idea" to be implemented
    b. should not be long
    c. should be generic, that is it should be reusable (used in other code).
    d. Should be **readable**!!!
    e.
4. general format:
    a. name
    b. return type (before the name), value returned must match that type.
    c. the params, each with a type.
5. Function invocation starts the function:
    a. passes arguments
    b. return value (if there is one) is captured by an assignment
        i. how many things can you return?
        ii. how can you get around that?
        iii.

7. Chars and Strings

1. How do you indicate a character? What is it the type name?
2. What is Unicode? What is it important?
3. Do you know the basic char functions (isalnum, isalpha, isdigit, islower, etc.)
    a. good for a cheat sheet
4. Strings are an STL class, what does that mean? Are they a base type (like int)?
    a. what can you do with an STL object you cannot with a base type?
5. Strings store a sequence of what?
6. Input of strings:
    a. what's the difference between a cin>>a_str and getline(cin, a_str)?
7. What are two ways to index the characters of a string? What is the difference?
8. Any difference between .length() and .size() methods?

9. What does string::npos represent (how do we use it in code)?
    a. what are those :: things again?
10. What is a size_type for a container? How does it differ from an int?
11. I love range based for loops. Can you write one?
    a. what does that auto mean in a range based for?
    b. what does a range for print during each iteration on a string? Why?
12. Couple ways to construct an STL object (including a string). Know what they mean.
    a. string s;
    b. string s(10, '=');
    c. string s{'a','b','c'};
    d. string s1(s2);
13. some string methods:
    a. substr
    b. push_back
    c. find (know how this one works, how do you know you found something).
        i. variations here. Do you know them (cheat sheet).

Second midterm topics start here

8. More on Functions

1. Function definition:
    a. receives parameters
    b. defines the operation of the function
2. Function declaration:
    a. what is that (what is required)
    b. how can that be used to support multiple-people programming?
        i. what kind of file?
3. Argument to Parameter matching:
    a. first argument of the invocation is matched to the first parameter of the definition.
    b. each function defines a local scope.
    c. the value stored in the parameter is the value of what was stored in the argument.
    d. can you pass out a reference/pointer to a local var as a return?
        i. compiler error?
        ii. run-time error?
4. What happens during the pass of argument to parameter?
    a. unless we say otherwise, it is a copy?
    b. what does an & in the parameter list mean? How does it affect the calling argument?
    c. what does a `const` in the parameter list mean?
5. What does it mean that a function is "overloaded"
    a. what are the elements taken into consideration?

    b. what complicates overloading
6. Function Templates
    a. what is a function template?
        i. is it a function?
    b. why are templates so important in C++?
    c. How are types selected in a template?
        i. can you force it? If so, how

9. Streams

1. How does a stream relate to a device (what is it's job)?
2. do you what the extraction operator and insertion operator do?
3. The fail state of cin means what?
    a. how to fix (more complicated than you think, look at the slides).
4. What does cin.ignore do? Why do we need it?
5. If you use while (cin >> some_str) as a phrase, under what circumstances does the while loop end?
6. What's the difference between cout and cerr?
    a. could we change that?
7. output formatting can be challenging. Do you know all the little settings (pg 757)?
8. What does endl really do (two things).
9. how can you open a file (two ways)
10. what's the difference between ifstream and ofstream?
11. Do the standard ostream operators work with file streams?
12. ***Don't worry*** about all the file modes, there for your own benefit
13. I love string streams. What are they? What are they good for?
14. What are the two types?
15. How do you set one, get a string out of one (what's the method)?\
16. Can you format a string stream with stuff like boolalpha, setprecision etc?
17. ***Don't worry*** about the seek and tell stuff (for your benefit)

10. Vectors & Iterators

1. What does it mean that the STL containers are "templated"
    a. are they all templated? Name one that isn't.
    b. can I mix and match types in a single container?
2. Can you declare a vector of a particular type (do you know the syntax)?
3. What is the difference between capacity and size?
    a. what does this have to do with memory management in a vector?
4. vector ops
    a. `capacity`
    b. `size`
    c. `empty`
    d. `push_back`
    e. `pop_back`
    f. `[]`
    g. `==, <`

    h. `front`
    i. `back`
5. if you us a range-based for a vector, what type of element comes out on each iteration?
6. What is an iterator?
    a. what is guaranteed about iterators (that perhaps is not guaranteed for ops like [])?
7. For an STL container, what does `.begin()` and `.end()` return?
    a. what is their type? What does it depend on?
8. how do you dereference an iterator?
9. Can you write code for three different ways to iterate over a vector?
10. We talked about pointer arithmetic. What is special about adding something to a pointer?
    a. what does pointer++ mean in terms of elements in a container like a vector?
11. What's the difference between (*ptr)+1 and *(ptr+1)?
12. Do you understand the translation between range-based for and an iterator approach? What about the stuff on & and const &
13. Do you know:
    a. `begin(), end()`
    b. `cbegin(), cend()`
    c. `rbegin(), rend()`
    d. `crbegin(), crend()`
14. Do you understand the iterator classes:
    a. forward iterators
    b. bi-directional iterators
    c. random-access iterators

11. Generic Algorithms

1. What's the big deal about generic algorithms? What can they do that other algorithms cannot? Can you list them?
    a. how do iterators play into this?
    b. does the underlying type of the container matter?
2. There are too many of these to memorize so let's focus on the ones we covered in the slides explicitly.
    a. `accumulate`
        i. what operation is assumed?
        ii. why is the initial value important?
        iii. can you change it from + operator?
        iv. can you write your own function? Can you do it?
    b. `find, search`
        i. what's up with algorithms ending in _if?
        ii. what kind of function is required here?
        iii. what's the difference between find and search?
    c. `copy, transform`
        i. copy assumes what about the target of the copy

ii. how can you get around that problem (`back_inserter`, what does it do)?
iii. what is an `ostream_iterator`, what is it used for?
1. it needs a template, remember that.

`d. sort`

i. what operator is assumed to work on the container?
1. can you write your own (can you?)

3. We did lots of examples, make sure you can follow those.

12. Associative Containers

1. Why is the `pair` type important in a discussion about STL maps?
   a. how do you specify a `pair` (couple of ways)?
   b. how do you access the two parts?
2. Maps allow only a single occurrence of a key in the map
   a. any restriction on values?
3. What kind of iterators in a map?
   a. what comparison operator are you restricted to in iteration?
4. Maps are "ordered". What does that mean?
   a. what part is ordered?
   b. what operator is assumed to set the order?
5. You don't `push_back` into a map, rather you `insert`
   a. what gets inserted?
   b. what gets returned in an insert on an ordered map?
6. What's weird about subscript `[ ]` operations on maps?
   a. subscript value is the key, yielding the value (like Python)
   b. what should you do instead to determine if a key is in the map?
7. For a map iterator, what does `itr->first` mean?
   a. what does the iterator point to?
   b. what is an equivalent expressions?
   c. remember, keys are `const`, values are not
8. Sets are similar to maps
   a. single occurrence only of element
   b. elements, not pairs, in a set
   c. element is const, can't change it.
   d. are there methods for sets for union, intersection etc? If not, where are they?
      i. what do they require?
9. What is interesting about multiset, multimap
   a. multiple key instances
   b. no subscript [ ], find instead
10. unordered containers are not ordered (surprise). ***Don't worry*** about these so much
11. ***don't worry*** about lambdas.

13. Classes I

1. Property 1, Aggregation

2. struct/class looks like a type
    a. you define what the contents are: data members
    b. you can also define methods: function members
3. Can you write/read a struct?
    a. don't forget the weird ; at the end!
4. declaration of a struct/class in a header file
    a. usually has an associated implementation/definition file
    b. declaration only requires the types
    c. what's different about includes for your own headers?
5. When a variable of the new struct is declared, that variable has stored in it:
    a. all the data member elements
    b. can be access via methods using the function members (methods)
6. Access is using a "dot" call:
    a. Clock my_c; my_c.hours;
    b. remember the -> operator, very useful here
7. How can you tell a member function from a regular, every day function. Two things!
    a. part of the struct
    b. called using dot call
8. What is that :: thing mean again?
    a. how is it important for the member implementations?
9. Special variable with the name `this`
    a. what type is it
    b. who sets it
    c. how is it used in a method (do you have to do anything special)
        i. what is a "naked data member"?

14. Classes 2

1. Abstraction, 2nd property
    a. why is this really, really important?
    b. how do we provide abstraction (the interface)
        i. what file is the interface declaration?
2. What is a constructor (what does it do)
    a. do we have to provide one (what are the rules)?
    b. do they have a return type (why or why not)?
    c. what is a default constructor?
        i. what is default initialization?
    d. can a constructor be overloaded?
        i. what does that mean for a constructor?
3. C++ shortcuts:
    a. initializer list. Done in the header. Can you read/write one?
        i. what does inlining mean?
    b. what does =default mean?
4. What is the difference between "to conversion" and "from conversion"?
    a. which one is easier (we know how to do)?
    b. what does it require?

5. Is conversion a good idea?
    a. what is the difference between implicit and explicit conversion? what keyword can you use to prevent implicit conversion?