CSE 232, 1st Day

Spring 2018, Dr. Bill Punch

Breakdown

410 students strong split into 16 sections.

ACTUARIAL	3
APL ENGR S	10
ASTROPHYS	2
BCH/BMB/BT	2
BIO LAB SC	1
BUS PREF	3
BUS-ADMIT	3
CHEM EGR	7
CHEMISTRY	2
CMPTR EGR	50
CMPTR SCI	153
COMMUNICAT	1
COMPTL MTH	12
CRIM JUST	2
ECONOMICS	6
EGR-NO PRF	3
ELECTR EGR	11
ENGLISH	1
EXP ARCH	2
FINANCE	5
FOOD I MGT	2
GENOMICS	2
GEOG INFO	3
GEOLOG SCI	1
GERMAN	1
GRAPH DSGN	1
HI SCH GST	4
HISTORY	1
HUMAN BIOL	1
LB ASTRO	1
LB GEONOM	1
LB MATH	1
LB ZOOLOGY	1
LYMAN B	11
MANAGEMENT	1
MATHEMATIC	24
MECH EGR	25
MEDIA INFO	8
MTH ADVNC	5
NEUROSCI	1
NO PREF	5
PHILOSOPHY	1
PHYSICS	17
PLANT BIOL	1
SPLY C MGT	2
STATISTICS	10

38% Comp Sci
12% Comp Egr
60% Engineering

Demographics

38% Sophomore
34% Juniors
18% Seniors!

Freshman	39
Sophomore	154
Junior	138
Senior	75

About the Course

- Previous programming experience is *expected*.
  - We are going to go much faster than 231 because you know the basics.
  - Some of you tested in from Java, still likely faster
  - Some of you took C++ before, we'll go farther and faster

Things along the way

Besides C++/STL, you will learn

- Working in a Linux environment
- Command line terminal work
- Debugging (hopefully)

My (and now your) goals

Three parts:

1. Learn C++ Syntax
2. Learn STL containers and algorithms
3. Make our own containers

Material

Course material is available on the Web at http://www.cse.msu.edu/~cse232

Everything is basically on this site, or at the very least a link to what you want.

Check here first!

Three important things

1. your labs start **this week**!
2. The lab is EB 3345
   1. your new home
3. the course is ***absolutely full***,
   1. If you want to swap, do it on piazza

Questions? Piazza not Email

You should have received an invitation to connect to Piazza, where we provide group forums for questions

- ask your questions there, so everyone can see the answer
- we monitor and answer questions
  - you can answer questions too!

Contacting Punch

I have an email but if you want to get my attention for 232 questions send to the following:

cse232@cse.msu.edu

I'll monitor that email channel separately and will try to answer questions within 24 hours.

Material/Text

- C++ Primer, 5th Edition (Lippman, Lajoie, Moo).
  - Great C++ reference, you use it a lot
  - pretty cheap
  - you will use it for other courses
- Online examples
- Slides

A new book

I'm trying to write a new book for the course. I have proofs and I might share, it depends on how much time we get.

Computer Projects

- There will be 11 computer projects -- roughly one per week, and constitute 45% of the course grade.
- **Late computer projects are not accepted**.
- Projects are typically due each Monday before midnight (except for exam weeks)

Auto grading

- We will use the website Mimir (class.mimir.io) for automatic grading
  - Invitation is on the front page of the CSE232 D2L page.
- Mimir provides tests for your code as well as a web interface for coding
  - Good way to start, but you need to find a better, local way as time progresses
- Your project grade will be based on the Mimir tests you pass

Pre-labs

- We will do pre-lab exercises on D2Lto emphasize the points being made in lab and for projects
- They are on D2L and are due each week, Thur before midnight
- They are graded by D2L
  - You can answer as many times as you like.

Labs

- There are required, weekly laboratory exercises.
  - Labs ***based on your section***
  - ***Cannot attend a different section***!!!
- To pass the course, a student ***can only miss two labs without penalty***
- Grading on labs is credit/no_credit.
- Collaboration on labs is ***encouraged***.

Labs

Students who ***miss more than two labs*** will have their ***final grade reduced by 0.5*** for each over the 2 limit lab missed.

- No excuses required for the labs you miss. Visit your parents, dentist appointment, sleep in, up to you
- ***No makeups for labs***. You can miss two. If you miss, then that is one, you have one left. Get it?

your responsibility

TAs record your lab attendance and your worksheet work on D2L.

Check it! No one can fix attendance months after the fact. Up to you.

Score breakdown, 1000 points total

- 1st Exam:
  - 150 pts (15% overall, 25% exam score)
- 2nd Exam:
  - 150 pts (15% overall, 25% exam score)
- Final:
  - 200 pts (20% overall, 33% exam score)
- Projects
  - total 11 projects for 450 points (45%)
- In class exercises
  - sum total of all exercises we do this semester will be worth 50 points (5%)

Grading Scale

4.0	900 -1000 points
3.5	850 - 899 points
3.0	800 - 849 points
2.5	750 - 799 points
2.0	700 - 749 points
1.5	650 - 699 points
1.0	600 - 649 points
0.0	0 - 599 points

Adjustments, or "The Curve"

We do not curve individual grades! Rather, we do the following. In the Python class, the 3.0/3.5 boundary was placed near the sum of exam medians plus 450, i.e. a student who was at the median on exams and had perfect projects was near the 3.0/3.5 cut.

- True last semester. Actually higher!

Weekly Work

We are trying to organize better to include the online folks. We will provide the work in weekly chunks. You will see the links on the website.

On-Campus Computing

- All students taking CSE courses get an account on EGR computers which they can access remotely or in the EGR labs third floor Engineering.
- The EGR computing labs are open 7x24 and your account is active on all the machines in all the labs.
- ***Lab00*** gets you set up for this!!!

On-Campus Help Room

- A help room is provided. Help room hours will be posted by the second week.
- The help room is overcrowded the day projects are due so do not expect extensive help at the last minute.
- We will be utilizing the HIVE in Wilson for the help room to deal with the load

How to work from home

FAQ (232 web page provides details):

1. x2go : provides a full desktop as a window in your laptop
   1. needs a decent network connection
   2. you connect to your CSE directory!
2. Mimir (also your handin)
3. Work on your own on your own machine (but beware of grading!!!!!)

Our Focus is C++11

In 2011 the latest standard for C++ came out, inventively named C++11, with some of C++14 and C++17

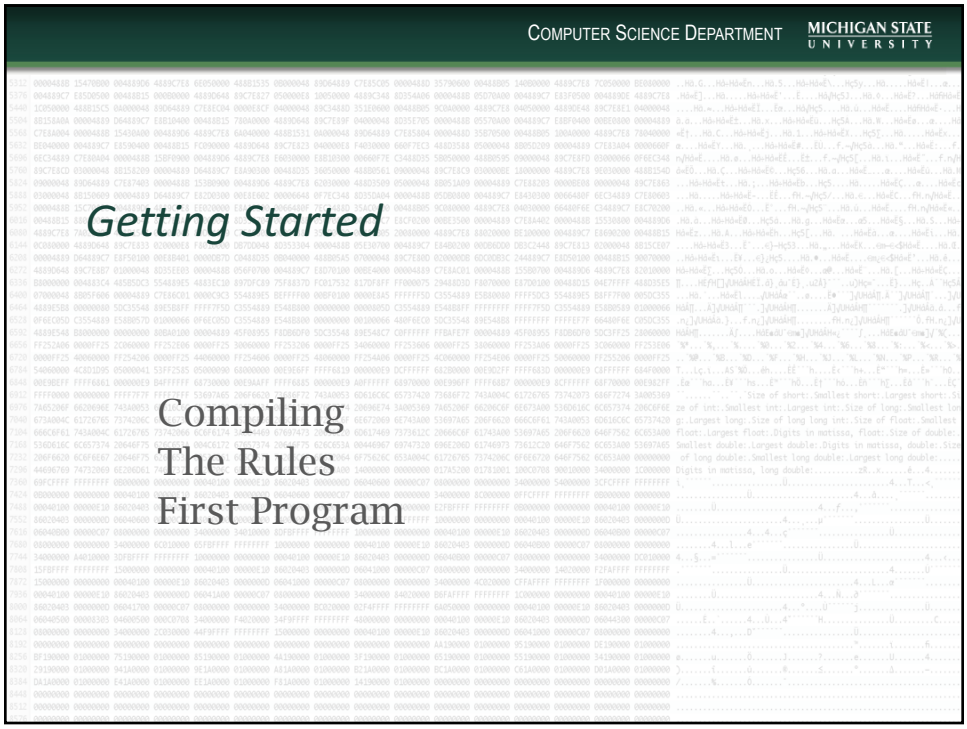
You will learn the latest standard:

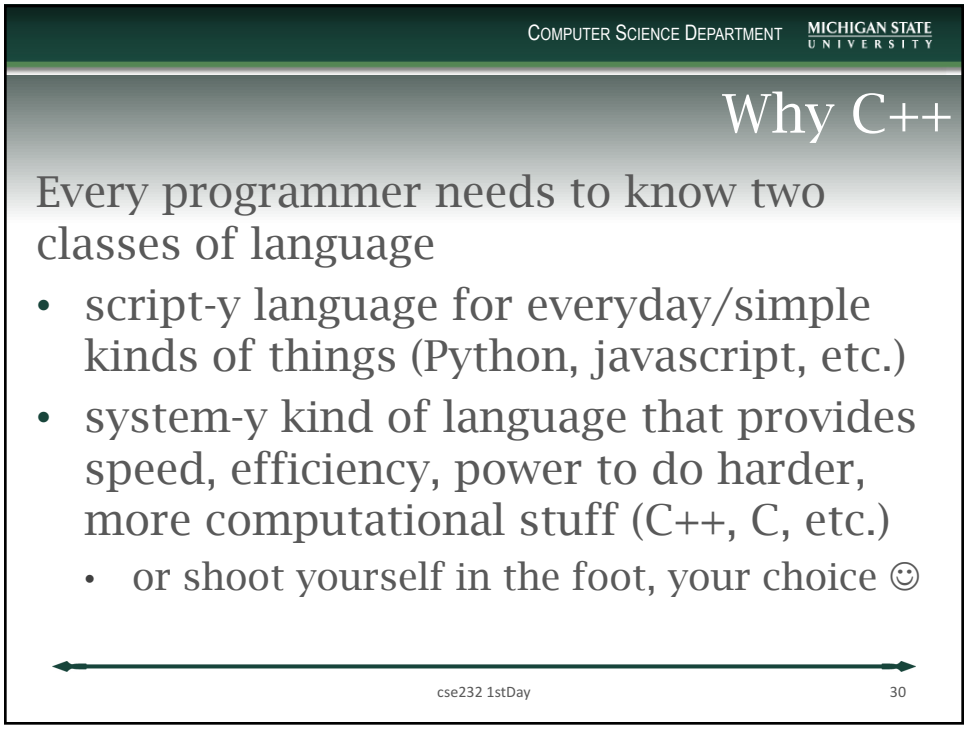
- it's easier to work with
- it makes C++ "better"
- amaze your friends and professors (they don't know this stuff!).

The STL

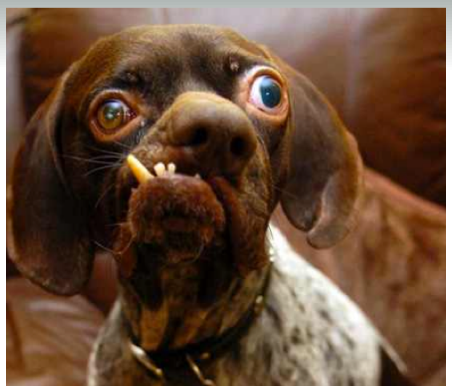
The Standard Template Library (STL) is your best friend. It does the work you don't have to, don't want to, can't do as well.

We will ***focus*** on using the STL. Use it whenever you can!


COMPUTER SCIENCE DEPARTMENT
MICHIGAN STATE UNIVERSITY
Getting Started
Compiling
The Rules
First Program


COMPUTER SCIENCE DEPARTMENT
MICHIGAN STATE UNIVERSITY
Why C++
Every programmer needs to know two classes of language
• script-y language for everyday/simple kinds of things (Python, javascript, etc.)
• system-y kind of language that provides speed, efficiency, power to do harder, more computational stuff (C++, C, etc.)
• or shoot yourself in the foot, your choice ☺
cse232 1stDay
30

C++, you'll grow to love it



Kind of of like an ugly dog. Looks bad at the beginning but you can grow to love it. Sort of.

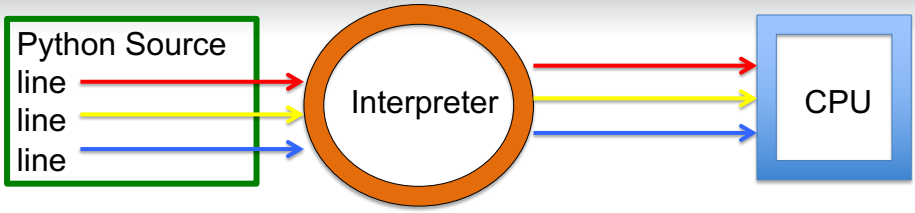
Some rules to help guide us

Provide some general rules that emphasize the difference between Python-like languages and C++

Rule 1. *You* do the work

In C++, you (the programmer) do the work. There is no interpreter to help you. Your code must stand on its own.

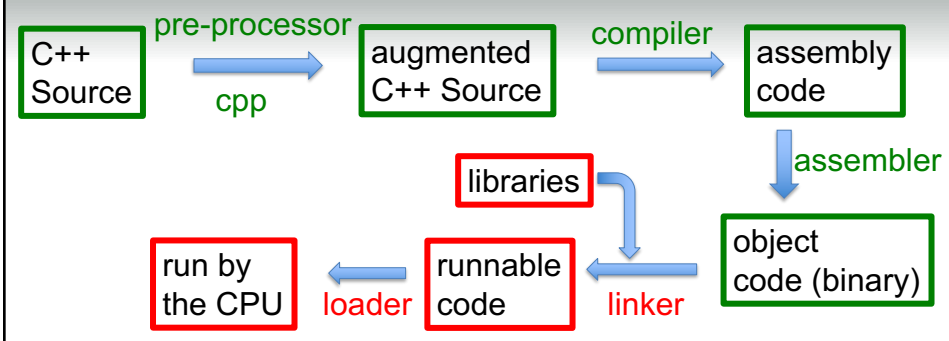
Interpreted Code



Each line of the source is processed and the interpreter runs that code. Interpreter is (well mostly) part of running the code

- does lots of book-keeping for you (yeah!)

Compiled Code



Green is compile, steps g++/clang++ takes

Red is run, the steps to run the code

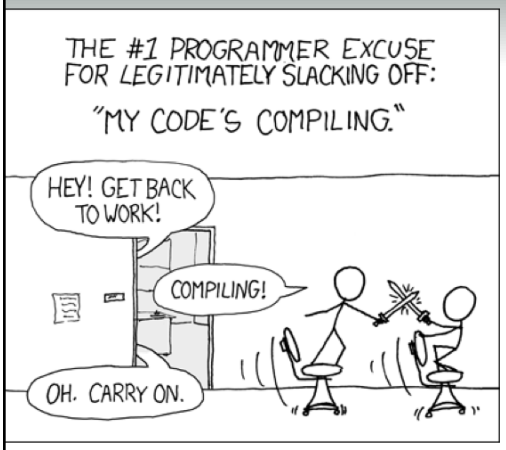
Compiled Code stands on its own

Compiled code has no interpreter-like program resident while it runs.

The compiled code (the object code) must do all the work that the interpreter might do:

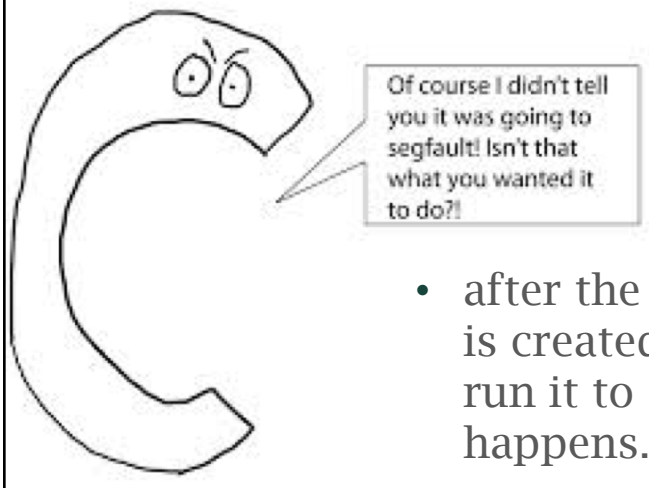
- memory creation/destruction
  - garbage collection

Compile time



- when the compiler software runs on your program creating an executable

Run time



- after the executable is created, we can run it to see what happens.

Compile Time Error

You did something that violates the "rules of C++"

The compiler thinks you're crazy, doesn't know what to do, flags the line that makes no sense and quits

No executable created!!!

Run time error

You followed the rules of C++ but you still screwed up

- an executable was created but it did something wrong/stupid/scary and you need to fix it
- could be logic, could be misuse (not violation) of C++, something

Standard Template Library helps

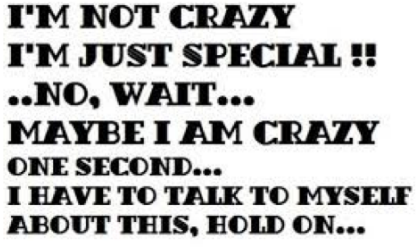
The Standard Template Library (STL) helps. It provides libraries, in particular containers and algorithms, that each know how to handle tasks like memory

STL also provides general algorithms that know how to manipulate containers

Rule 2

C++ is absolutely positively crazy about types. Not only are there types, but there are all kinds of modifiers to types.

at ***compile time***, C++ must know the types of data before it can successfully compile



Example

```
const map<string, vector<long>> const * m = &some_var;
```

green stuff is all part of a single type declaration.

- a constant pointer to a constant map of string to a vector of longs pointer

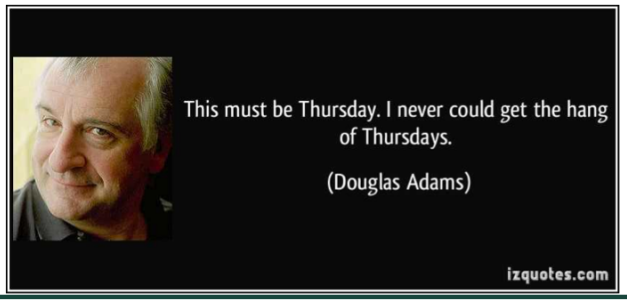
Pretty eh? You won't even blink by mid semester when you see that.

Declare your variables

- you have to declare your variable's type
- once declared, it only holds that type (or does something to what you try to stuff into that type)

Rule 2 is a hard rule!

Getting the hang of types, getting the types correct, can be maddening, especially after a scripting language which is so loose about types.



Rule 3: Syntax requires context

We have a couple of things working against us when it comes to C++ syntax:

- C++ tries to stay backwards compatible with languages like C.
- C++ continues to add features to the language that really help.
- There are only so many characters on a standard keyboard

Context, context, context

The result is the same symbols get used to mean completely different things, and the only way you can figure that out is **in context**.

That is, the symbol is not enough to sort out what it means, you need to look at the local context (the code that is around it).

Example

```
long my_int = 123;
long &ref_int = my_int;
long *another_int = &my_int;
my_int = *another_int;
my_int = my_int * my_int;
```

Line 2, & means "a reference type"
Line 3, & means "the address of"
Line 3, * means "pointer type"
Line 4, * means "value pointer points to"
Line 5, * means multiplication

Rule 3 is still pretty hard

You just have to be careful. "Look around" any symbol to figure out what it means.

Surprisingly, you get used to it after awhile (but then you can get used to anything).

Rule 4: compiler error messages can be cryptic
Rule 4.1: compile all the time

C++ has a well earned reputation that its error messages can be terrifying.

A single syntax error can generate 10's of lines of error.

- important info is the line number.

I forgot a single "<", should be <<

```
cout << "Size of short:"<<sizeof(short)< endl;
```

cse232 1stDay

COMPUTER SCIENCE DEPARTMENT MICHIGAN STATE UNIVERSITY

The important part

Line 19, character 43

```
>g++ -std=c++11 2.1-basicTypes.cc
2.1-basicTypes.cc: In function 'int main(int,
char**)':
2.1-basicTypes.cc:19:43: error: no match for
'operator<' in '(& std::operator<<
<std::char_traits<char> >((* & std::cout),
((const char*)"Size of short:")))-
>std::basic_ostream<_CharT, _Traits>::operator<<
<char, std::char_traits<char> >(2ul) < std::endl'
```

... lots of other crap ...

The MOST IMPORTANT THING!

To keep the confusion down, you should ***compile all the time***. That is:

- write a line
- recompile
- write the next line
- recompile
- …

If not, you suffer. Plain and simple.

Simple Programs

First Program

```
#include <iostream>
/*
 wfp, 7/8/13
 hello world program
*/

int main(){
  std::cout << "Hello World"; // output
}
```


edit compile run

This is the cycle. edit-then-compile

1. compile the code
   1. if successful, run the compiled code
   2. if failure, edit and fix errors, recompile
2. Once compiled, run the compiled code
   1. if it doesn't do what you want, edit and recompile
3. Finished

includes

To use aspects of the library system, you must include the definitions into the system (like import in Python)

Table A.1 (866-870) lists many of the elements and their associated include file

sign indicates the pre-processor

```
#include<iostream>
```

is a pre-processor statement. This one is for I/O processing. It **does not end** in a semicolon because is part of the pre-processor, not part of the C++ language

- What you include goes between `< >`
- No `.h` or `.hpp` at the end of the `include`

Comments, two kinds

`/* ... */`

Anything between those symbols are ignored. Can be multiline

`// ...`

From that point of the line to the end is a comment. One line only

main

A runnable program requires a specific function named `main`

This function is what will start the program when the compiled code is linked and loaded.

First Program

```
#include <iostream>
/*
 wfp, 7/8/13
 hello world program
*/
int main ()
{
  std::cout << "Hello World";
}
```

return type

func name

argument list (now empty)

begin and end of a block

{ } means a block (mostly)

Again, remember that context is important. Curly brackets have multiple meanings:

In the context of a function or a control statement, curly brackets mean **block**:

a **sequence** of statements that fill in for a location where a single statement goes

Indentation means nothing

For the Python folks, indentation means **nothing**. Below is the previous program as a single line which is valid (but sucks):

```
#include <iostream>/* wfp, 7/8/13 hello world program*/ int main(){std::cout << "Hello World"; // output}
```


Indentation, like comments, for reading

Like comments, though C++ doesn't care, we as programmers care

We need to make our code readable

First Program

```
#include <iostream>
/*
 wfp, 7/8/13
 hello world program
*/
int main(){
  std::cout << "Hello World"; //output
}
```

std namespace

cout stream

std

When you do an `include` you insert definitions into your program. But those new elements need to be referenced by the namespace they are in. For now, we are always in the `std` (standard) namespace

- `std` is short for standard
  - Yes, I know it is an unfortunate acronym.

The lazy typist

No group of human beings are lazier typists than programmers.

- short acronyms instead of full names
- short variable names

Need to find a balance between readability and typing.

scope resolution operator

To differentiate namespaces, you include the namespace name followed by `::`

- two colons are the *scope resolution operator*
- `std::cout` means `cout` in the standard namespace

Streams

When you `include<iostream>`, you get three streams for your use:

- `std::cout` (short for console output)
- `std::cin` (short for console input)
- `std::cerr` (short for console error)

Must, somehow, indicate the namespace

We will talk about alternatives, but somehow you have to indicate what namespace the stuff you `include` shows up as.

Weird errors otherwise

First Program

```
#include <iostream>
/*
 wfp, 7/8/13
 hello world program
*/

int main(){
  std::cout << "Hello World"; //output
}
```

insertion operator

Output

To move information from your program to output, you use the << (insertion) operator

- take info and place it in the output
- is a binary operator
  - **returns the stream being used**
- outputs either strings (between " ") or the value in a C++ variable

" " vs ' '

"abc" is a string type (sequence of characters)

'a' is a character type, a ***single*** character

First Program

```
#include <iostream>
/*
 wfp, 7/8/13
 hello world program
*/
int main(){
  std::cout << "Hello World"; //output
}
```

return type

no semi here

semicolon yes

semicolons

C++ uses the `;` (semicolon) to terminate a line. Not all lines require them:

- expressions require them
- declarations require them
- blocks **do not** require them
- we'll get the hang of where they go.
- A good editor will help with missing ; and similar issues.

Style guide

Style guide

We select a style for writing our programs so:

- they are more readable (very important)
- we have a system amongst ourselves so we can agree on format
- be religiously picky about how we do things ☺

Google

Why not use Google's style guide

http://google-styleguide.googlecode.com/svn/trunk/cppguide.xml

They have a very complete style guide

- when in doubt, consult

Not a problem for the first project, will be later

Variable names: rules and style

Variable names Rules:

- only digits, letters and underline allowed
- can't start with digit
- case matters
- namespace matters

Variable name Style:

- "lower with under": like Python, lower case letters joining multiple words by underline.
- meaningful/readable!

Comments

- comment at the top of the file. Name, date, what the file is about
- variable names shouldn't require comments (descriptive names)
- functions should have comments (input and output, plus what it does)
- "*If it was hard to write, it will be hard to read*"
  - comment the hard parts

First Program

```
#include <iostream>
/*
 wfp, 7/8/13
 hello world program
*/

int main(){
  std::cout << "Hello World"; // output
}
```