Introduction to C++

Dr. Robert Lowe

Division of Mathematics and Computer Science
Maryville College

Maryville COLLEGE

Outline

Outline

Your Repository

1. Check your email. Look for an invite to a repository named:
   `cs1-fall2019-`*`username`*

Your Repository

1. Check your email. Look for an invite to a repository named: `cs1-fall2019-`*`username`*
2. Accept the invitation.

Your Repository

1. Check your email. Look for an invite to a repository named: `cs1-fall2019-`*`username`*
2. Accept the invitation.
3. Look at the repository on GitHub.

Your Repository

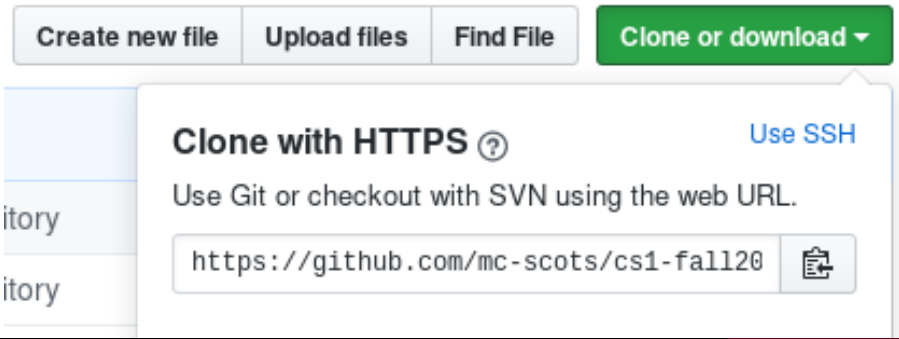
1. Check your email. Look for an invite to a repository named: `cs1-fall2019-`*`username`*
2. Accept the invitation.
3. Look at the repository on GitHub.

4. Log in to cs.maryvillecollege.edu via ssh.

Your Repository

1. Check your email. Look for an invite to a repository named: `cs1-fall2019-`*`username`*
2. Accept the invitation.
3. Look at the repository on GitHub.

4. Log in to cs.maryvillecollege.edu via ssh.
5. On GitHub, click “Clone or download” and select “https”.



Your Repository (ctd.)

6. Copy the https URL to your clipboard.

Your Repository (ctd.)

6. Copy the https URL to your clipboard.
7. Type the following command in your ssh shell (pasting the URL where specified)

```
git clone «Paste URL Here»
```

Your Repository (ctd.)

6. Copy the https URL to your clipboard.
7. Type the following command in your ssh shell (pasting the URL where specified)
   `git clone «Paste URL Here»`
8. Change into your `cs1-fall2019-username` directory.
   HINT: Use tab completion! Type `cd cs1` and then press the tab key. It saves time!

Your Repository (ctd.)

6. Copy the https URL to your clipboard.
7. Type the following command in your ssh shell (pasting the URL where specified)
   `git clone «Paste URL Here»`
8. Change into your `cs1-fall2019-username` directory. **HINT**: Use tab completion! Type `cd cs1` and then press the tab key. It saves time!
9. Look around in your newly cloned repository. This is where you will do all of your work!

Basic `git` Commands

Basic Pattern: `git` *`command args`*

`git clone` *`url`* Clone a repository to your current directory.

Basic `git` Commands

Basic Pattern: `git` *`command args`*

`git clone` *`url`* Clone a repository to your current directory.

`git pull` Pull changes from GitHub. (Do this at the beginning of each class meeting!)

Basic `git` Commands

Basic Pattern: `git` *`command args`*

`git clone` *`url`* Clone a repository to your current directory.

`git pull` Pull changes from GitHub. (Do this at the beginning of each class meeting!)

`git add` *`filename`* Add a file to the list of files tracked by git.

Basic `git` Commands

Basic Pattern: `git` *`command args`*

`git clone` *`url`* Clone a repository to your current directory.

`git pull` Pull changes from GitHub. (Do this at the beginning of each class meeting!)

`git add` *`filename`* Add a file to the list of files tracked by git.

`git add -A` Add all untracked files to the git repository.

Basic `git` Commands

Basic Pattern: `git` *`command args`*

`git clone` *`url`* Clone a repository to your current directory.

`git pull` Pull changes from GitHub. (Do this at the beginning of each class meeting!)

`git add` *`filename`* Add a file to the list of files tracked by git.

`git add -A` Add all untracked files to the git repository.

`git commit -a` Commit all changes to the local repository. (Do this at the end of every major change.)

Basic `git` Commands

Basic Pattern: `git` *`command args`*

`git clone` *`url`* Clone a repository to your current directory.

`git pull` Pull changes from GitHub. (Do this at the beginning of each class meeting!)

`git add` *`filename`* Add a file to the list of files tracked by git.

`git add -A` Add all untracked files to the git repository.

`git commit -a` Commit all changes to the local repository. (Do this at the end of every major change.)

`git push` Push all changes to GitHub. (Do this at the end of every work session.)

Your Workflow

Each time you sit down to work:

1. Login on MCCS via ssh.

Your Workflow

Each time you sit down to work:

1. Login on MCCS via ssh.
2. Change into your `cs1-fall2019-`*`username`* directory.

Your Workflow

Each time you sit down to work:

1. Login on MCCS via ssh.
2. Change into your `cs1-fall2019-`*`username`* directory.
3. `git pull`

Your Workflow

Each time you sit down to work:

1. Login on MCCS via ssh.
2. Change into your `cs1-fall2019-`*`username`* directory.
3. `git pull`
4. Do some rather taxing and stressful programming.

Your Workflow

Each time you sit down to work:

1. Login on MCCS via ssh.
2. Change into your `cs1-fall2019-`*`username`* directory.
3. `git pull`
4. Do some rather taxing and stressful programming.
5. Periodically (and at the end): `git add -A`

Your Workflow

Each time you sit down to work:

1. Login on MCCS via ssh.
2. Change into your `cs1-fall2019-`*`username`* directory.
3. `git pull`
4. Do some rather taxing and stressful programming.
5. Periodically (and at the end): `git add -A`
6. Periodically (and at the end): `git commit -a`

Your Workflow

Each time you sit down to work:

1. Login on MCCS via ssh.
2. Change into your `cs1-fall2019-`*`username`* directory.
3. `git pull`
4. Do some rather taxing and stressful programming.
5. Periodically (and at the end): `git add -A`
6. Periodically (and at the end): `git commit -a`
7. At the end of your work session: `git push`

Your Workflow

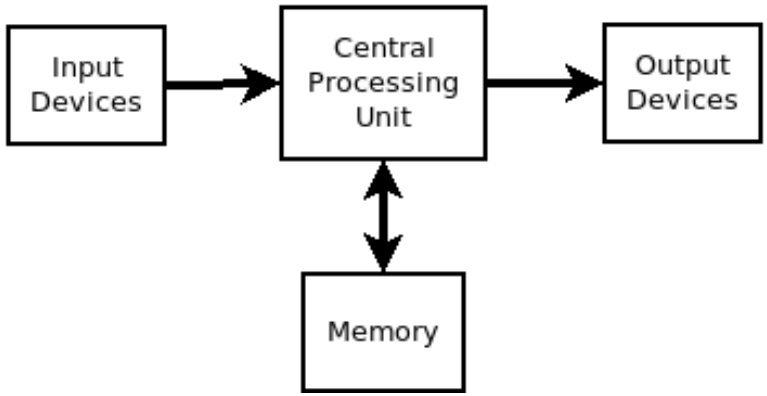
Each time you sit down to work:

1. Login on MCCS via ssh.
2. Change into your `cs1-fall2019-`*`username`* directory.
3. `git pull`
4. Do some rather taxing and stressful programming.
5. Periodically (and at the end): `git add -A`
6. Periodically (and at the end): `git commit -a`
7. At the end of your work session: `git push`
8. Contemplate coding until the blessed hour arrives when you can resume your work.

Outline

Anatomy of a Computer

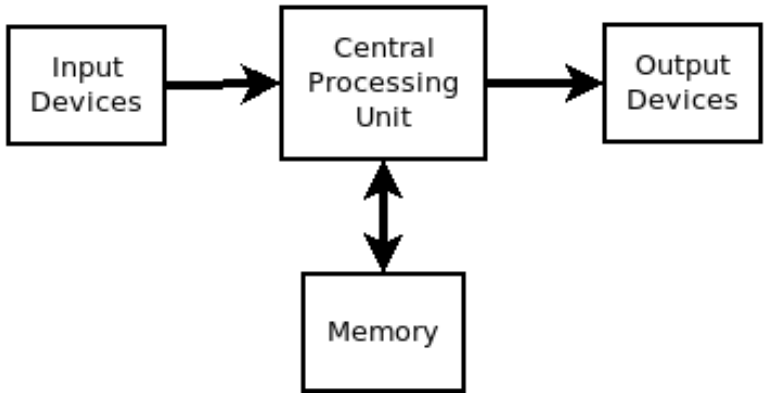
- A **computer** is a device which executes programs.



Von-Neumann Architecture

Anatomy of a Computer

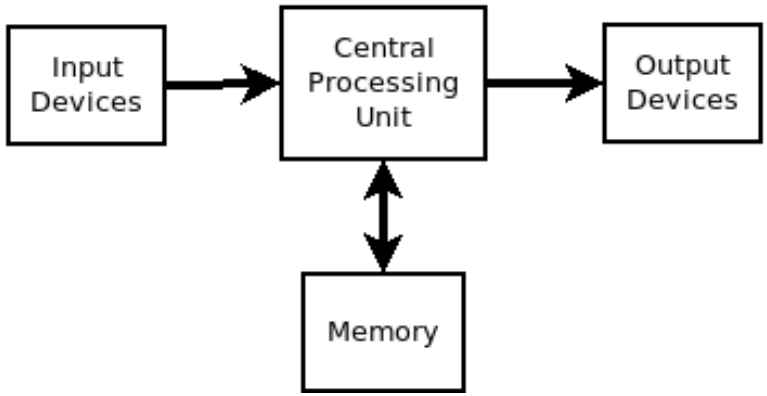
- A **computer** is a device which executes programs.
- The most common computer architecture is the Von-Neumann Architecture.



Von-Neumann Architecture

Anatomy of a Computer

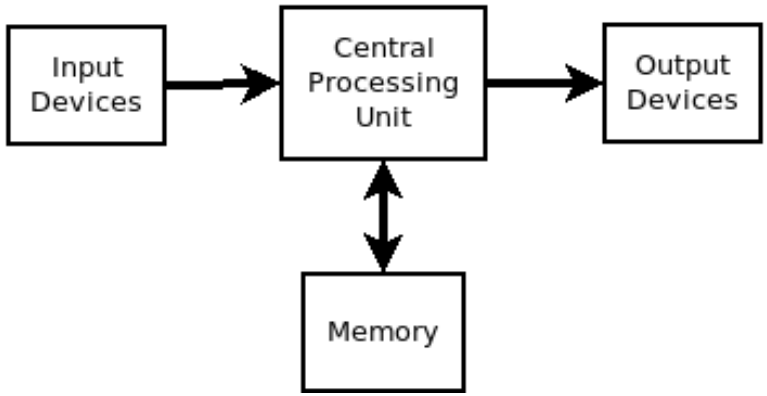
- A **computer** is a device which executes programs.
- The most common computer architecture is the Von-Neumann Architecture.
- Program code and data are stored in the same memory.



Von-Neumann Architecture

Anatomy of a Computer

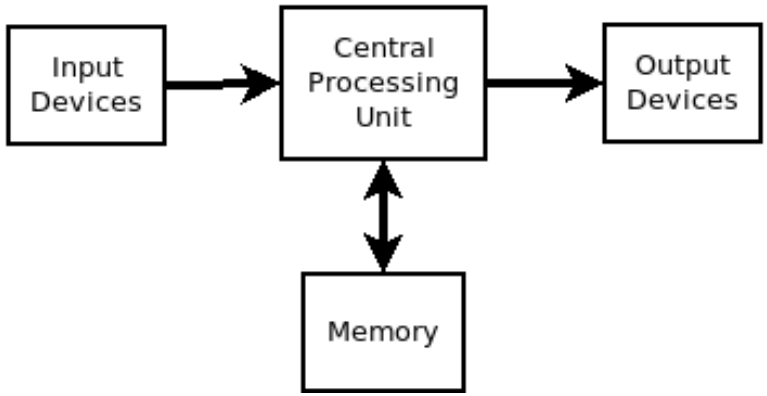
- A **computer** is a device which executes programs.
- The most common computer architecture is the Von-Neumann Architecture.
- Program code and data are stored in the same memory.
- **CPU** Executes very simple instructions.



Von-Neumann Architecture

Anatomy of a Computer

- A **computer** is a device which executes programs.
- The most common computer architecture is the Von-Neumann Architecture.
- Program code and data are stored in the same memory.
- **CPU** Executes very simple instructions.
- Collectively, the components of a computer are called its **hardware**.



Von-Neumann Architecture

Program Representation & Translation

- The text typed by a programmer is called **source code**.

Program Representation & Translation

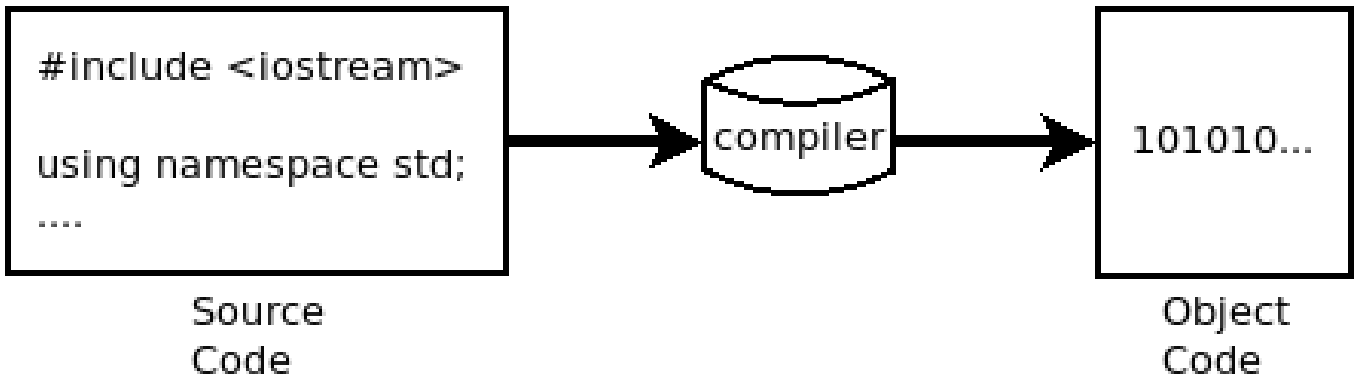
- The text typed by a programmer is called **source code**.
- The binary code executed by a computer is called **object code** or **machine code**.

Program Representation & Translation

- The text typed by a programmer is called **source code**.
- The binary code executed by a computer is called **object code** or **machine code**.
- In **interpreted** languages an **interpreter** directly executes source code.

Program Representation & Translation

- The text typed by a programmer is called **source code**.
- The binary code executed by a computer is called **object code** or **machine code**.
- In **interpreted** languages an **interpreter** directly executes source code.
- In **compiled** languages a **compiler** translates source code into object code.



A Program - In Machine Code

```
b8 04 00 00 00 bb 01 00 00 00 b9 25 10 40 00 ba
33 10 40 00 81 ea 25 10 40 00 cd 80 b8 01 00 00
00 31 db cd 80 68 65 6c 6c 6f 2c 20 77 6f 72 6c
64 0a 00 00
```

A Program - In Machine Code

```
b8 04 00 00 00 bb 01 00 00 00 b9 25 10 40 00 ba
33 10 40 00 81 ea 25 10 40 00 cd 80 b8 01 00 00
00 31 db cd 80 68 65 6c 6c 6f 2c 20 77 6f 72 6c
64 0a 00 00
```

- What does this do?

A Program - In Machine Code

```
b8 04 00 00 00 bb 01 00 00 00 b9 25 10 40 00 ba
33 10 40 00 81 ea 25 10 40 00 cd 80 b8 01 00 00
00 31 db cd 80 68 65 6c 6c 6f 2c 20 77 6f 72 6c
64 0a 00 00
```

- What does this do?
- Really, even this needs to have a bit of a wrapper to turn it into binary.

A Program - In Machine Code

```
b8 04 00 00 00 bb 01 00 00 00 b9 25 10 40 00 ba
33 10 40 00 81 ea 25 10 40 00 cd 80 b8 01 00 00
00 31 db cd 80 68 65 6c 6c 6f 2c 20 77 6f 72 6c
64 0a 00 00
```

- What does this do?
- Really, even this needs to have a bit of a wrapper to turn it into binary.
- Change into your `examples/01-Intro-C++` directory.

A Program - In Machine Code

```
b8 04 00 00 00 bb 01 00 00 00 b9 25 10 40 00 ba
33 10 40 00 81 ea 25 10 40 00 cd 80 b8 01 00 00
00 31 db cd 80 68 65 6c 6c 6f 2c 20 77 6f 72 6c
64 0a 00 00
```

- What does this do?
- Really, even this needs to have a bit of a wrapper to turn it into binary.
- Change into your `examples/01-Intro-C++` directory.
- `cat prognum.S`

A Program - In Machine Code

```
b8 04 00 00 00 bb 01 00 00 00 b9 25 10 40 00 ba
33 10 40 00 81 ea 25 10 40 00 cd 80 b8 01 00 00
00 31 db cd 80 68 65 6c 6c 6f 2c 20 77 6f 72 6c
64 0a 00 00
```

- What does this do?
- Really, even this needs to have a bit of a wrapper to turn it into binary.
- Change into your `examples/01-Intro-C++` directory.
- `cat prognum.S`
- `as prognum.S -o prognum.o`

A Program - In Machine Code

```
b8 04 00 00 00 bb 01 00 00 00 b9 25 10 40 00 ba
33 10 40 00 81 ea 25 10 40 00 cd 80 b8 01 00 00
00 31 db cd 80 68 65 6c 6c 6f 2c 20 77 6f 72 6c
64 0a 00 00
```

- What does this do?
- Really, even this needs to have a bit of a wrapper to turn it into binary.
- Change into your `examples/01-Intro-C++` directory.
- `cat prognum.S`
- `as prognum.S -o prognum.o`
- `ld prognum.o -o prognum`

A Program - In Machine Code

```
b8 04 00 00 00 bb 01 00 00 00 b9 25 10 40 00 ba
33 10 40 00 81 ea 25 10 40 00 cd 80 b8 01 00 00
00 31 db cd 80 68 65 6c 6c 6f 2c 20 77 6f 72 6c
64 0a 00 00
```

- What does this do?
- Really, even this needs to have a bit of a wrapper to turn it into binary.
- Change into your `examples/01-Intro-C++` directory.
- `cat prognum.S`
- `as prognum.S -o prognum.o`
- `ld prognum.o -o prognum`
- `./prognum`

A Program - In Assembly

```
.text
.globl _start

_start:
            movl    $4, %eax
            movl    $1, %ebx
            movl    $msg, %ecx
            movl    $msgend, %edx
            sub     $msg, %edx
            int     $0x80

            movl    $1, %eax
            xorl    %ebx, %ebx
            int     $0x80
msg:    .string "hello, world\n"
msgend: .byte 0
```

A Program - In Assembly

```
.text
.globl _start

_start:
            movl    $4, %eax
            movl    $1, %ebx
            movl    $msg, %ecx
            movl    $msgend, %edx
            sub     $msg, %edx
            int     $0x80

            movl    $1, %eax
            xorl    %ebx, %ebx
            int     $0x80
msg:    .string "hello, world\n"
msgend: .byte 0
```

- Make sure you are in `examples/01-Intro-C++`.

A Program - In Assembly

```
.text
.globl _start

_start:
            movl    $4, %eax
            movl    $1, %ebx
            movl    $msg, %ecx
            movl    $msgend, %edx
            sub     $msg, %edx
            int     $0x80

            movl    $1, %eax
            xorl    %ebx, %ebx
            int     $0x80
msg:    .string "hello, world\n"
msgend: .byte 0
```

- Make sure you are in `examples/01-Intro-C++`.
- `cat hello.S`

A Program - In Assembly

```
.text
.globl _start

_start:
            movl    $4, %eax
            movl    $1, %ebx
            movl    $msg, %ecx
            movl    $msgend, %edx
            sub     $msg, %edx
            int     $0x80

            movl    $1, %eax
            xorl    %ebx, %ebx
            int     $0x80
msg:    .string "hello, world\n"
msgend: .byte 0
```

- Make sure you are in `examples/01-Intro-C++`.
- `cat hello.S`
- `as hello.S -o hello.o`

A Program - In Assembly

```
.text
.globl _start

_start:
            movl    $4, %eax
            movl    $1, %ebx
            movl    $msg, %ecx
            movl    $msgend, %edx
            sub     $msg, %edx
            int     $0x80

            movl    $1, %eax
            xorl    %ebx, %ebx
            int     $0x80
msg:    .string "hello, world\n"
msgend: .byte 0
```

- Make sure you are in `examples/01-Intro-C++`.
- `cat hello.S`
- `as hello.S -o hello.o`
- `ld hello.o -o hello`

A Program - In Assembly

```
.text
.globl _start

_start:
            movl    $4, %eax
            movl    $1, %ebx
            movl    $msg, %ecx
            movl    $msgend, %edx
            sub     $msg, %edx
            int     $0x80

            movl    $1, %eax
            xorl    %ebx, %ebx
            int     $0x80
msg:    .string "hello, world\n"
msgend: .byte 0
```

- Make sure you are in `examples/01-Intro-C++`.
- `cat hello.S`
- `as hello.S -o hello.o`
- `ld hello.o -o hello`
- `./hello`

A Program - In Python

```
print "hello, world"
```

A Program - In Python

```
print "hello, world"
```

- Make sure you are in `examples/01-Intro-C++`.

A Program - In Python

```
print "hello, world"
```

- Make sure you are in `examples/01-Intro-C++`.
- `cat hello.py`

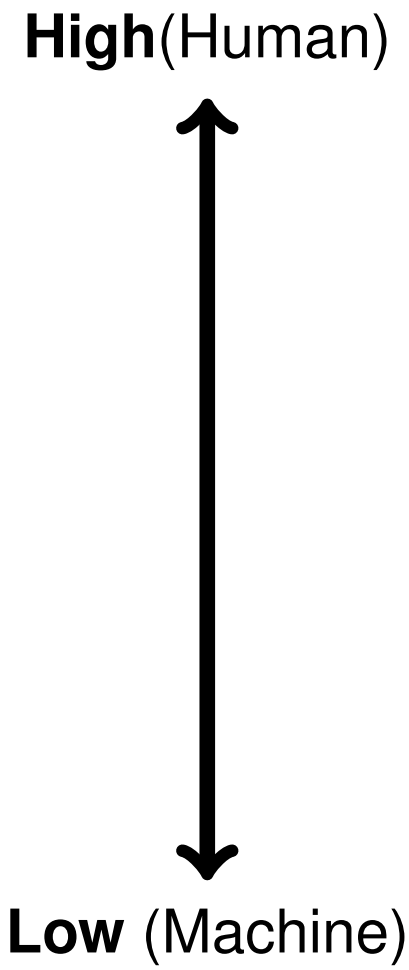
A Program - In Python

```
print "hello, world"
```

- Make sure you are in `examples/01-Intro-C++`.
- `cat hello.py`
- `python hello.py`

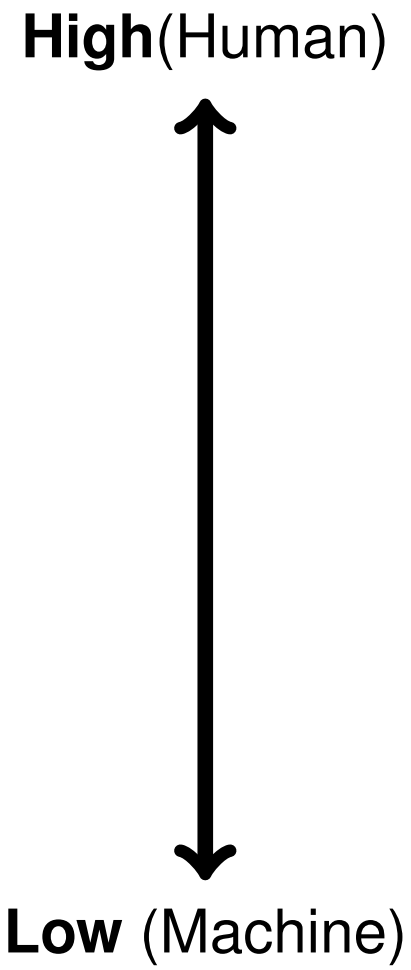
Levels of Abstraction

- (Too) Many programming languages exist.



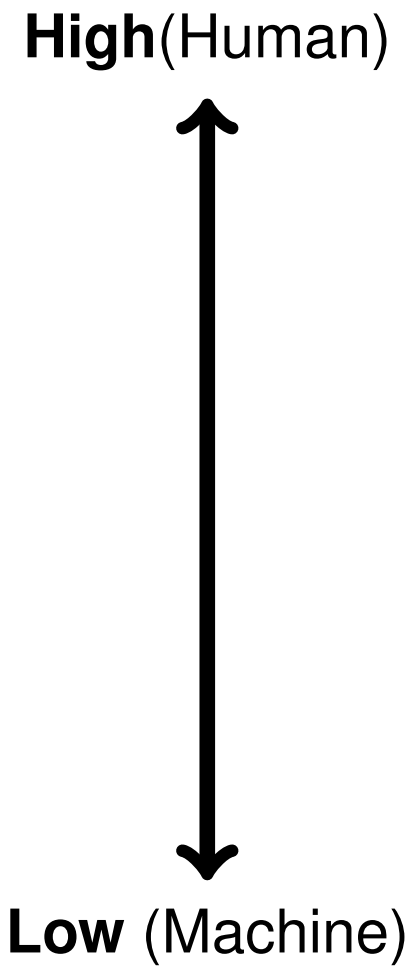
Levels of Abstraction

- (Too) Many programming languages exist.
- Each language is designed for some set of purposes.



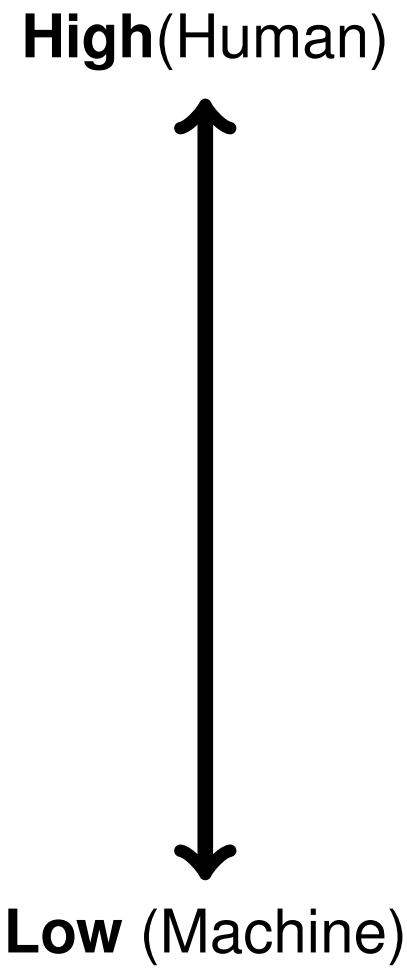
Levels of Abstraction

- (Too) Many programming languages exist.
- Each language is designed for some set of purposes.
- Usually, humans desire a fair degree of **abstraction**.



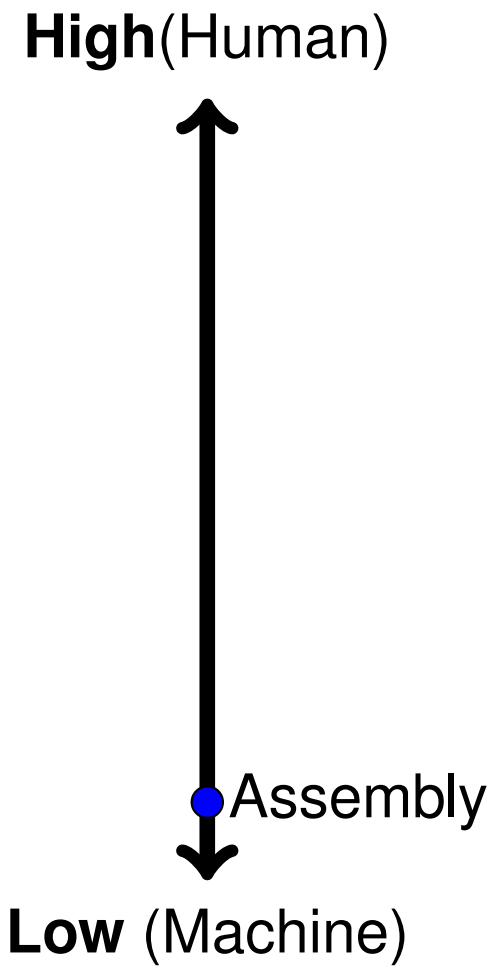
Levels of Abstraction

- (Too) Many programming languages exist.
- Each language is designed for some set of purposes.
- Usually, humans desire a fair degree of **abstraction**.
- A programming language's level of abstraction is the degree to which it hides details from the programmer.



Levels of Abstraction

- (Too) Many programming languages exist.
- Each language is designed for some set of purposes.
- Usually, humans desire a fair degree of **abstraction**.
- A programming language's level of abstraction is the degree to which it hides details from the programmer.



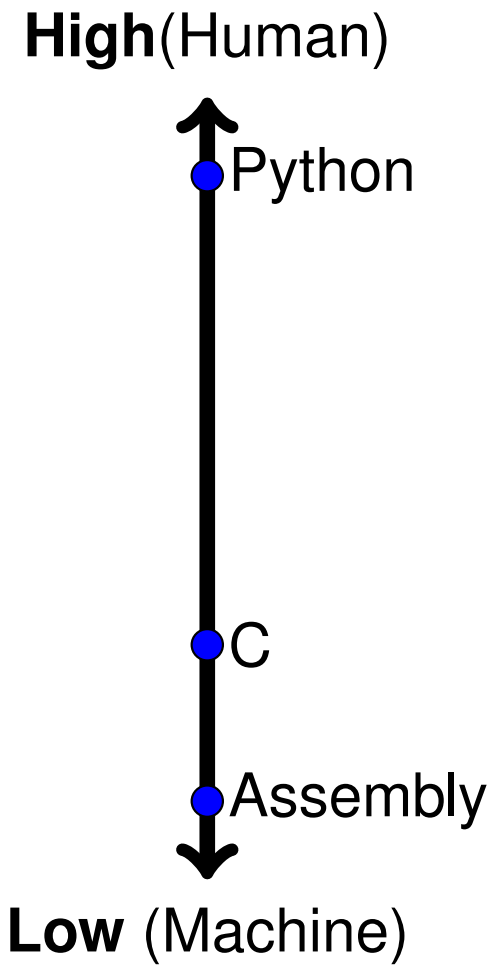
Levels of Abstraction

- (Too) Many programming languages exist.
- Each language is designed for some set of purposes.
- Usually, humans desire a fair degree of **abstraction**.
- A programming language's level of abstraction is the degree to which it hides details from the programmer.

Maryville COLLEGE

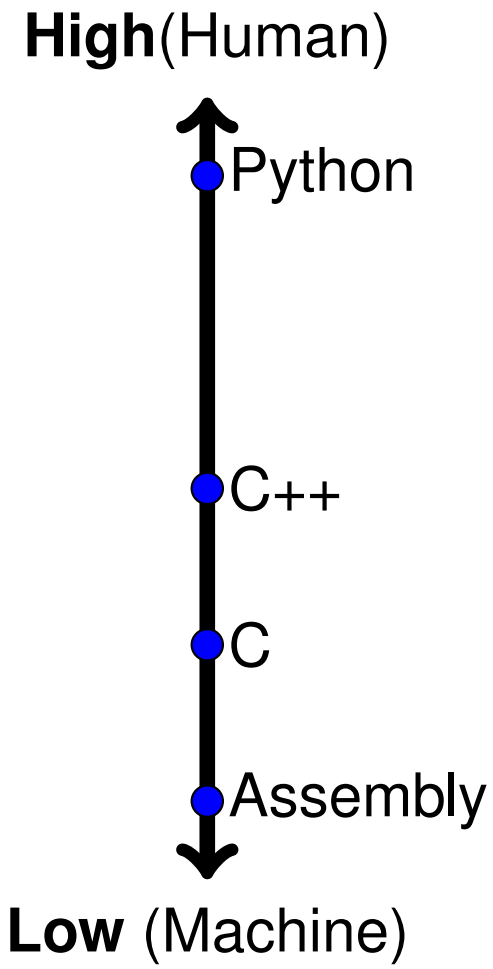
Levels of Abstraction

- (Too) Many programming languages exist.
- Each language is designed for some set of purposes.
- Usually, humans desire a fair degree of **abstraction**.
- A programming language's level of abstraction is the degree to which it hides details from the programmer.



Levels of Abstraction

- (Too) Many programming languages exist.
- Each language is designed for some set of purposes.
- Usually, humans desire a fair degree of **abstraction**.
- A programming language's level of abstraction is the degree to which it hides details from the programmer.



The C++ Programming Language

Source: `https://en.wikipedia.org/wiki/Bjarne_Stroustrup`

The C++ Programming Language

- Created at Bell Labs by Bjarne Stroustrup as a successor to the C programming language.

Source: `https://en.wikipedia.org/wiki/Bjarne_Stroustrup`

The C++ Programming Language

- Created at Bell Labs by Bjarne Stroustrup as a successor to the C programming language.
- Adds Object Oriented Programming to C.

Source: `https://en.wikipedia.org/wiki/Bjarne_Stroustrup`

The C++ Programming Language

- Created at Bell Labs by Bjarne Stroustrup as a successor to the C programming language.
- Adds Object Oriented Programming to C.
- Compiled, Mid-Level language.

Source: `https://en.wikipedia.org/wiki/Bjarne_Stroustrup`

The C++ Programming Language

- Created at Bell Labs by Bjarne Stroustrup as a successor to the C programming language.
- Adds Object Oriented Programming to C.
- Compiled, Mid-Level language.
- Ported to virtually all modern platforms.

Source: `https://en.wikipedia.org/wiki/Bjarne_Stroustrup`

The C++ Programming Language

- Created at Bell Labs by Bjarne Stroustrup as a successor to the C programming language.
- Adds Object Oriented Programming to C.
- Compiled, Mid-Level language.
- Ported to virtually all modern platforms.
- Low level access with high level facilities.

Source: `https://en.wikipedia.org/wiki/Bjarne_Stroustrup`

The C++ Programming Language

- Created at Bell Labs by Bjarne Stroustrup as a successor to the C programming language.
- Adds Object Oriented Programming to C.
- Compiled, Mid-Level language.
- Ported to virtually all modern platforms.
- Low level access with high level facilities.
- You get to explore a bit of all worlds when learning C++!

Source: `https://en.wikipedia.org/wiki/Bjarne_Stroustrup`

Outline

Creating The `week2` Directory

1. Change to your `~/cs1-fall2019-`*`username`* directory.

Creating The `week2` Directory

1. Change to your `~/cs1-fall2019-`*`username`* directory.
2. `cd labs`

Creating The `week2` Directory

1. Change to your `~/cs1-fall2019-`*`username`* directory.
2. `cd labs`
3. `mkdir week2`

Creating The `week2` Directory

1. Change to your `~/cs1-fall2019-`*`username`* directory.
2. `cd labs`
3. `mkdir week2`
4. `cd week2`

Create `hello.cpp`

Using the text editor of your choice, enter the following into a file titled `hello.cpp`.

```
#include <iostream>

using namespace std;

int main()
{
    cout << "hello, world" << endl;
}
```

Compile `hello.cpp`

1. `g++ hello.cpp -o hello`

Compile `hello.cpp`

1. `g++ hello.cpp -o hello`
2. Correct any errors you may have encountered.

Compile `hello.cpp`

1. `g++ hello.cpp -o hello`
2. Correct any errors you may have encountered.
3. Once it compiles error free, proceed.

Compile `hello.cpp`

1. `g++ hello.cpp -o hello`
2. Correct any errors you may have encountered.
3. Once it compiles error free, proceed.
4. `./hello`

Push Your Work to GitHub

1. `git add hello.cpp`

Push Your Work to GitHub

1. `git add hello.cpp`
2. `git commit hello.cpp -m 'Added hello.cpp'`

Push Your Work to GitHub

1. `git add hello.cpp`
2. `git commit hello.cpp -m 'Added hello.cpp'`
3. `git push`

Push Your Work to GitHub

1. `git add hello.cpp`
2. `git commit hello.cpp -m 'Added hello.cpp'`
3. `git push`
4. Go to your GitHub repository and verify that your file is in your `labs/week2` folder.