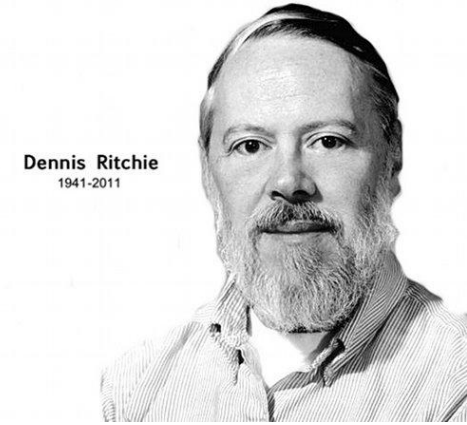
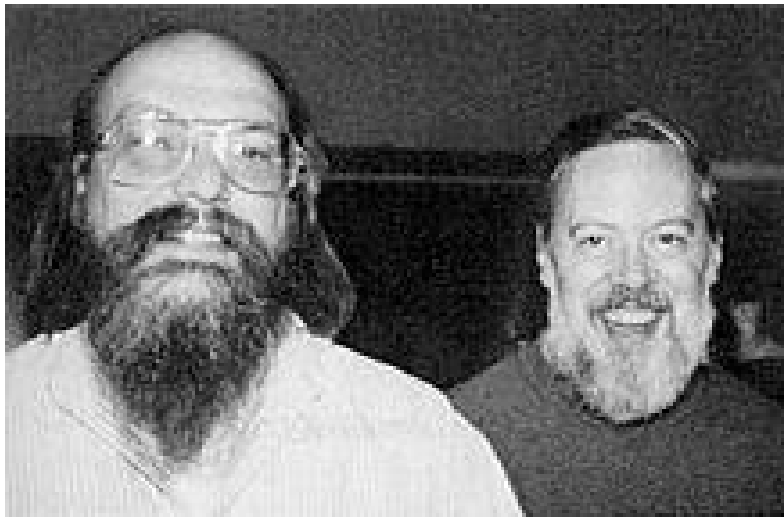
Introduction to C

C - A History

- Derived from B (Basic)
  - Visual Basic is also derived from B



- Released by Dennis Ritchie in 1973
  - Also co-wrote Linux with Ken Thompson



Why Use C?

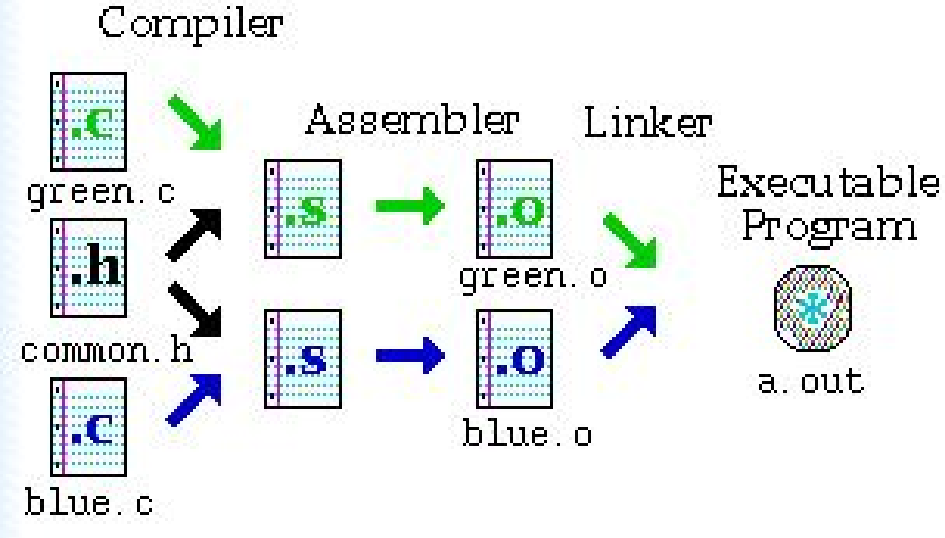
- It's fast: http://benchmarksgame.alioth.debian.org/u64q/performance.php?test=nbody
- Many large-scale programs (and other programming languages) are written in it
  - Linux
  - Python
  - Most compilers
  - Etc...

Why Different Programming Languages?

- Each designed to optimize a certain task
  - Python: Ease of typing
  - Java: Integrated with web interfaces
  - R: Statistical Computing
  - Swift: Integration with Apple products

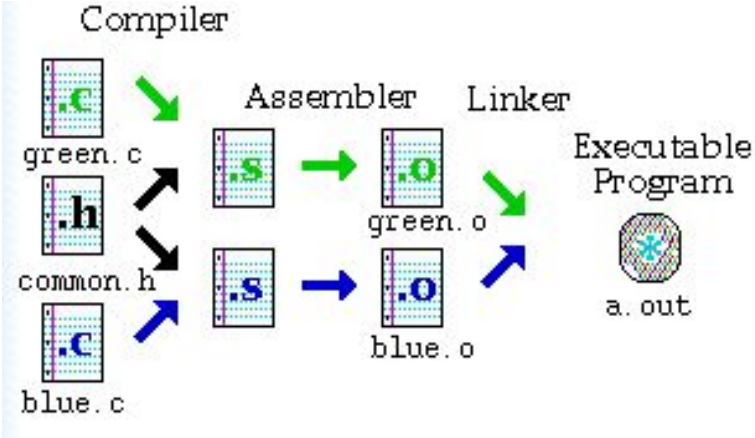
How Do These Programs Actually Run?

- Code is *compiled*
  - Translates human-written code (*source code*) into binary (*machine code*)



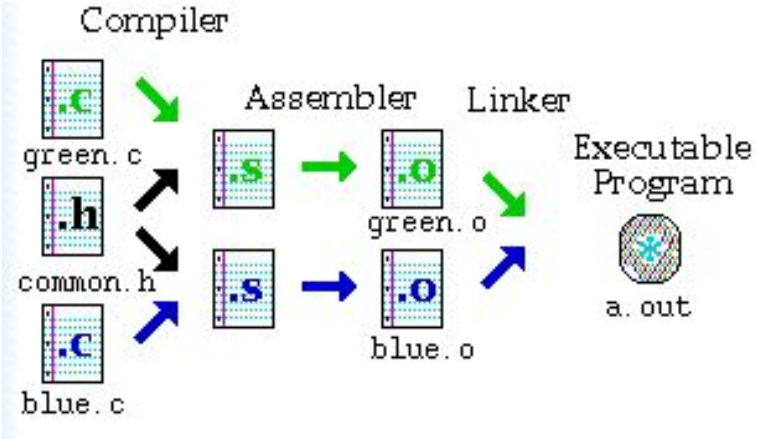
How Do These Programs Actually Run?

- Compilation: Step 1 = Preprocessing
  - Essentially organizes code into easy-to-process files (from .c and .h files to .s))



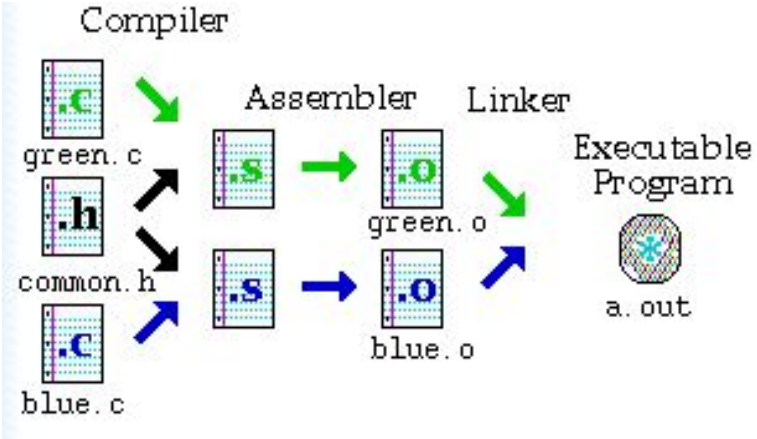
How Do These Programs Actually Run?

- Compilation: Step 2 = Translation
  - The *Compiler* translates preproccessed code into object code - gives syntax errors



How Do These Programs Actually Run?

- Compilation: Step 3 = Linking
  - The *Linker* takes the object code and combines all the code into an executable program



First C Program

```
>> nano hello.c

/*Use ctrl^o to save*/

/*Use ctrl^x to exit*/
```

More Nano commands: http://staffwww.fullcoll.edu/sedwards/Nano/UsefulNanoKeyCommands.html

```
/*This is a comment*/

#include <stdio.h>

int main(void) {

    printf("Hello, world!\n");

    return 0;

}
```

Compilation

- To run preprocessor and compiler:

```
>> gcc -Wall hello.c -c -o hello.o
```

- To run linker:

```
>> gcc hello.o -o hello
```

- To execute program

```
>> ./hello
```

Compilation - Easy Way

- To run preprocessor, compiler, and linker:

```
>> gcc -Wall hello.c -o hello
```

- To execute program

```
>> ./hello
```

C Standards

Comments: Between /* and */

Ignored by processors

Does not have to be written in code

```
/*This is a comment*/

#include <stdio.h>

int main(void) {

    printf("Hello, world!\n");

    return 0;

}
```

C Standards

Preprocessor directives: begin with #include

Include background code that allows commands to run

<stdio.h> includes the code for the *printf* command

```
/*This is a comment*/

#include <stdio.h>

int main(void) {

    printf("Hello, world!\n");

    return 0;

}
```

C STANDARDS

Int main(void): File header

REQUIRED IN EVERY C PROGRAM

Without this, the code will not run

Serves as the “doorway” to the code

```
/*This is a comment*/

#include <stdio.h>

int main(void) {

    printf("Hello, world!\n");

    return 0;

}
```

C STANDARDS

{ } : brackets show where the code is placed

Code outside brackets will give a compiler error

```
/*This is a comment*/

#include <stdio.h>

int main(void) {

    printf("Hello, world!\n");

    return 0;

}
```

C Standards

printf(“Hello, world!\n”);

Printf command: Prints the text inside the parentheses to the screen

\n: Newline character

```
/*This is a comment*/

#include <stdio.h>

int main(void) {

    printf("Hello, world!\n");

    return 0;

}
```

C Standards

printf(“Hello, world!\n”);

\n: Newline character. Equivalent of hitting the enter key in a Text Editor

```
/*This is a comment*/

#include <stdio.h>

int main(void) {

    printf("Hello, world!\n");

    return 0;

}
```

C STANDARDS

printf(“Hello, world!\n”);

; : Semicolon determines when the line of code ends

MUST BE INCLUDED AT THE END OF EVERY LINE - gives a compiler error if it does not exist

```
/*This is a comment*/

#include <stdio.h>

int main(void) {

    printf("Hello, world!\n");

    return 0;

}
```

C Standards

return 0;

Tells the *int main(void)* function to terminate

Stops the code

Technically can be omitted: But is required for this class

```
/*This is a comment*/

#include <stdio.h>

int main(void) {

    printf("Hello, world!\n");

    return 0;

}
```