What Is A Program?

- Programs are instructions for a computer, which tell it precisely how to behave
- Programs include Microsoft Windows, Microsoft Visual C++, Emacs, UNIX kernel, and many more
- These are all very, very complex. Our programs will be much simpler. But keep in mind that big things grow from little things...

But Exactly, What IS a Program?

- Typically refers to one of two things:
  - Source Code
  - Executable
- Source code is a human readable set of instructions for a computer. It is in a specific language (C, C++, Java, Pascal, Fortran, Basic, etc.)
  - Independent of machine
- An executable is a set of computer readable instructions
  - Machine dependent

Outline

- Theory: Edit - Compile - Link - Run
- Application: "Hello World"
- Practice: Using C Compilers
- Exercise: Activity 1.1

- Theory: Variables
- Application: Integers
- Application: Arithmetic
- Exercise: Activity 1.2

Subsections

- Theory
  - General theory -- "The art of computer programming"
  - Applies equally well to any language
  - About 1/2 the course
- Application
  - Apply the theory to C
  - C specific functionality
- Practice
  - Specific information about compilers, etc.

Theory

Programs are instructions for a computer, which tell it precisely how to behave. Programs include Microsoft Windows, Microsoft Visual C++, Emacs, UNIX kernel, and many more. These are all very, very complex. Our programs will be much simpler. But keep in mind that big things grow from little things...
Edit - Compile - ...

△ So how do we get from source code to machine code (executable)?
△ Edit
  □ Using a text editor, type the source code file
△ Compile
  □ Translate source code into an intermediate stage, called an object file

... - Link - Run

△ Link
  □ Different object files are combined together into a final executable
△ Run
  □ Invocation of the program. Machine code is loaded into memory and the computer begins following orders
△ Build
  □ Compile and link

Stairway To Heaven

Edit    Compile    Link    Run

edit    vi    emacs    notepad
Microsoft Visual C++
gcc
cc

"Hello World"
Our First C Program

#include <stdio.h>

int main(void)
{
  printf("Hello World\n");
  return 0;
}

Overview of "Hello World"

△ Output is:
  c:\temp> hello.exe
  Hello World
  c:\temp>

△ Let's go through the program:

#include <stdio.h>

△ Include directive: Tells the compiler to include support for input / output functions

Kenneth W. Flynn (7-12)
10/19/98
Through the World...

- `int main(void)`

- C programs are made up of functions.
  - We’ll learn how to write functions later
  - `main` is the first function called when a program is run
  - `int` and the set of parens afterwards have a particular meaning, which we’ll discuss later as well
  - For now, always declare `main` this way, and put your code inside it.

Where's The Beef?

- `printf("Hello World\n");`
  - This is the "meat" (or veggie burger) of the program
    - Does the actual work of printing the text
    - `printf` is a function
    - It is part of the `stdio` library (hence the `#include <stdio.h>`)
    - It’s first argument is a string (i.e. "Hello!")

Your Compiler Is a Block-head

- `{`
  - Braces define a `Block`. A block is a grouping of statements in a program
  - We’ll see more of blocks later on as well.
  - This particular brace marks the beginning of the program

Backslash N? A martial arts move, maybe?

- `
` is called an escape sequence
  - Escape sequences are used in strings to produce special characters that we cannot type
  - Two commonly used escape sequences are
    - `\n` : Prints a newline character
    - `\t` : Prints a tab character

Finishing Up...

- `return 0;`
- `}`
  - All functions return a value. For now, `main` should always return 0
  - Always end your programs like this, at least for a while
  - Note that this ENDS your program

Practice
Using Compilers

- Two kinds of compilers
  - IDE's
    - Visual C++
    - The Company Formerly Known As Borland's C++
    - Turbo C
  - Command Line
    - gcc
    - cc

Using Command Line Compilers

- gcc -Wall -ansi -pedantic hello.c
- Somewhat tricky
- Produces a file called a.out
- a.out is your executable
- Type "a.out" to run your program
- Available on all RPI UNIX workstations

Using Microsoft Visual C++

- Can you say Nuclear Weapon to kill a fly?
- Directions are in Activity 1

Activity 1.1

- Write a program to print the following:
  Welcome to C Programming!
  Congratulations on your first program!
- When you've completed the assignment, have myself or the TA check you off
**Variables**

- Computer Programs operate on some "data".
- This can be information from the user, from devices (like hard disks), or computed by the program.
- Regardless of its source, data needs to be stored somewhere by the program.
- Spaces in memory reserved for holding data are called *variables*.

"Dizzy to Variable..."

- Variables are stored internally in some machine format.
- They have a certain size in memory, generally given in bytes.
- Lingo:
  - *Bit*: A single electrical value in a computer; 0 or 1.
  - *Nybble*: 4 Bits.
  - *Byte*: 8 Bits.
  - *Word*: Varies, typically 2 or 4 Bytes.

**Integers**

- Integers are variables that hold simple numbers, like 1, 2, 3, or -3.
- In C, we typically declare variables as follows:
  ```c
  int x;
  ```
  - `int` means the variable is of type "integer".
  - `x` is the name of the variable.
  - Note the semicolon!

**Variable Names in C**

- Case Sensitive!
- Must begin with a letter.
- May contain letters, numbers, or the underscore `_`.
- Examples:
  - `x`
  - `count`
  - `first_integer`
  - `int2`

**Arithmetic**

- Operations on Integers:
  - `=` sets a variable to a value.
  - `+` adds two values.
  - `-` subtracts two values.
  - `*` multiplies two values.
  - `/` divides two values, and then truncates the result.
  - `%` divides two values and returns the remainder.
  - `()` group operations.
Arithmetic (continued)

△ All of these are called *operators*
△ Operators always produce a value (even `=`)

### Arithmetic Examples

△ `int x, y, z;`
△ `x = 5;`
△ `y = 2;`
△ `z = x + 3; /* z is 8 */`
△ `z = 3 + x;`
△ `z = x * y; /* z is 10 */`
△ `z = x / y; /* z is 2 */`
△ `z = x % y; /* z is 1 */`
△ `z = x = 2; /* x is 2, z is 2 */`
△ `3 * 2; /* Valid, but silly */`

Precedence

△ As in algebra, different operators have different precedence
△ Table on p. 36 of your text; in order:
   △ ()
   △ Stuff inside is resolved with the same rules
   △ * / %
   △ + -
   △ =

### Precedence Example: (Stars Go First? Not Quite)

△ `x = (5 * 3 + 1) + 2 * 3`
△ `x = (15 + 1) + 2 * 3`
△ `x = (16) + 2 * 3`
△ `x = 16 + 6`
△ `x = 22`

△ Note that when there is a precedence tie, evaluation proceeds left to right

Example: The Black Adder

△ /* The Black Adder */

```
#include <stdio.h>

int main(void)
{
    int x, y, z;
    x = 3;
    y = 2;
    z = x + y;
    printf("Z is %d.\n", z);
    return 0;
}
```

### Black Adder Output

△ `~> a.out`
△ `z is 5.
~>`
**Comments**

- // and */ enclose a comment
- Anything in a comment is ignored by the compiler. In fact, it is removed during the compile step
- Comments provide hints to readers of your code about what it is doing
- **COMMENT YOUR CODE WELL!**

**More On printf**

- %d is a formatting directive
- Technically, the string passed to printf is called a format string.
- Format strings can include normal text (as we have saw earlier), escape sequences, and special format directives
- %d means: "print a number here"
- Ok, so it's not intuitive
- For each format directive, printf takes an additional argument -- what to put there, in order

**More printf Examples**

```c
x = 3;
printf("X = %d. Wow.\n", x);
X = 3. Wow.

y = 2;
printf("%d\t%d", x, y);
3 2
```

**Activity 1.2**

- Write a program with four variables: i, j, k, and l. Initialize them to 1, 2, 3, and 0, respectively. Set l = i + j. Print it. Set l to j multiplied by the sum of i and k. Print it.
- Modify the program to read i, j, and k from the user. Then compute the values above.