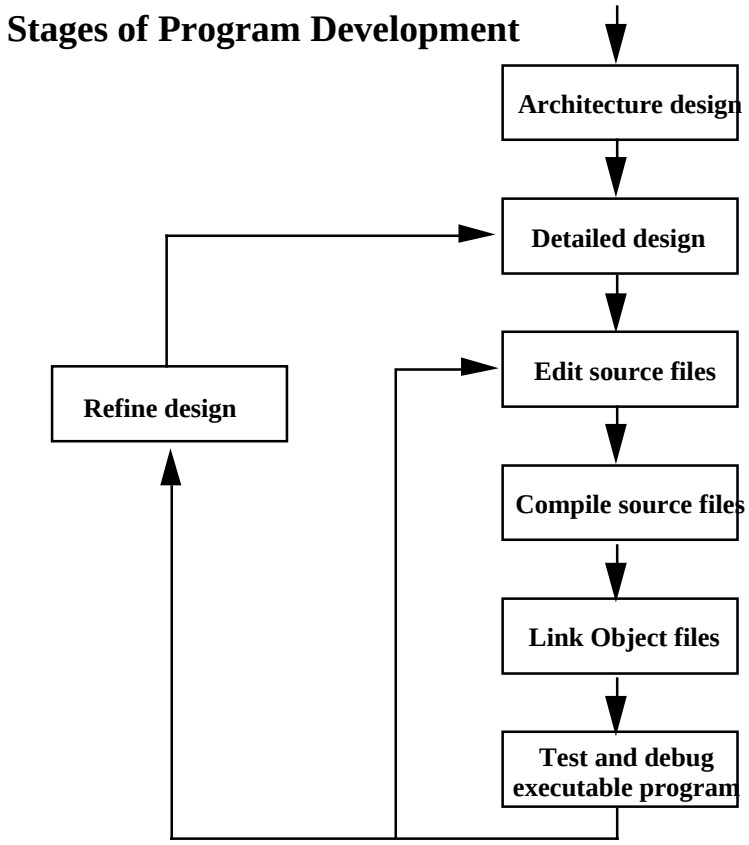
Crash Course in C

"A little learning is a dangerous thing." (Alexander Pope)

- The mechanics of program construction
- Structure of a C program
- Simple examples of C programs

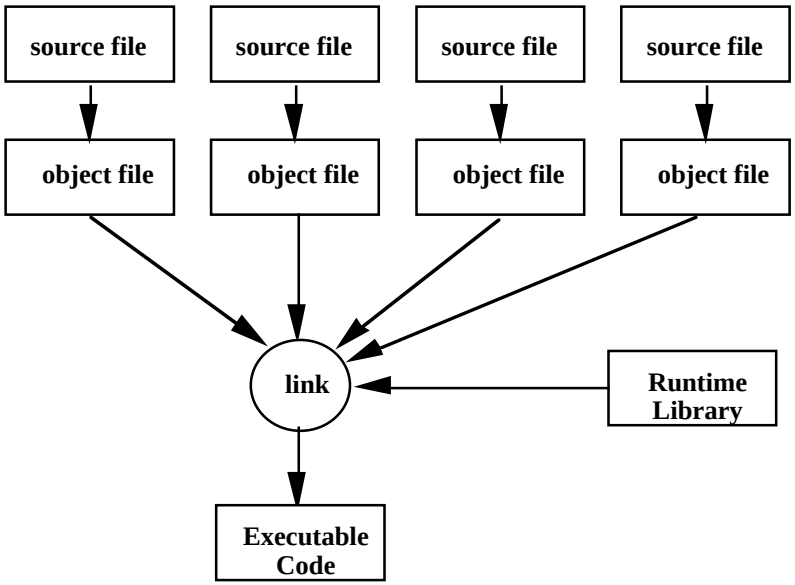
The Mechanics of Program Construction



Writing C Code

- Once the program has been designed, it needs to be written in a language that can be translated into a format that the computer can understand
  - English is a very loose language
  - Computers are very rigid machines
  - English description of the program must be converted to a computer language (C, Ada, Fortran, Pascal, Java, Basic, Assembler, etc.)
- Writing a C program involves creating and editing C language text files, called source files
- However, a computer can not execute, or run, a C source file

Creating an Executable Program



Compiling Source Files

- Once your program is written in C, it is ready to be translated into a machine-readable language
- A compiler translates C statements into machine statements
  - A compiler creates object code, which is an intermediary step between source code and final executable code
  - The compiler checks for syntax errors; e.g., Missing punctuation
  - The compiler performs simple optimization on your code; e.g., eliminate a redundant statement

Linking Object Files

- The linker links together all object modules to form an executable image of the program
- The output of the linker is an executable image, which can be loaded into memory and executed
- The linker resolves any references to library functions
  - If your program uses a library routine, like *sqrt*, the linker finds the object code corresponding to this routine and links it within the final executable image
- The linker is automatically invoked by the compiler

Loading Your Program

- The loader loads your program into the computer's memory
- On most systems, this is performed automatically by the operating system when you run the program
- Most embedded systems require you to explicitly run a loader program to get the program into memory

Mechanics of Program Construction - Review

- Let's review the steps involved in building a C program. This assumes that you have already designed the program and defined it using the steps above
  1. Use an editor to create/edit the program and any data files
     - Use meaningful names for files and variables; e.g., flightsim.c vs program3.c
  2. Type in your program, complete with parentheses, braces, semicolons
  3. Build (compile and link) your program. If compiler identifies errors (e.g., missing brace), edit your program and recompile.
  4. Execute your program.
     - Run your program using different test cases
     - If program does not execute properly, edit and return to step 3
  5. After program runs successfully, make printouts of source code and of test results.

Structure of a C program

- Program consists of one or more functions
- Function consists of header and body
  - Header contains preprocessor statements and declarations
  - Body, enclosed in {brackets} contains statements, each terminated by semicolon
- One function must be one called *main()*
  - The *main* function is where execution of the program begins
  - The *main* function begins at the line containing *main()* and ends at the closing brace on the last line of the source listing
  - The body of the *main* function contains one or more statements that describe the functionality of your program
- Program execution starts in *main*, and progresses, statement by statement, until the last statement in *main* is completed

Style of a C Program - Commenting and Indenting Your Code

- Properly commenting and indenting your code is an important part of programming
  - Enhance readability
  - Allow others to understand more quickly
  - Allow you to recall sooner
- Provide information at beginning of each source file that describes the code, date modified, by whom
- Intersperse comments within the code to explain intent
- Style guide on the class webpage. Use it!

Anatomy of a C Function

- A function, including the main function, contains the following elements
  - Function type
  - Function name
  - Left Parenthesis
  - Argument declarations*
  - Right parenthesis
  - Left curly bracket
  - Declarations*
  - C statements*
  - Right curly bracket

```
fcn-type fcn-name (arg-declarations)
{
        declarations;
        C-statements;
}
```

* = optional

A Simple Example

- Let's define a function that computes the square of the number 5
  1. File name: What shall we call the file? _____
  2. Function name: What shall we call the function? _____
  3. Function type: What type of value will the function return?
     _____
  4. Executable statements: Write the statement to perform the computation ___
  5. Return statement: Specify the output of the function _____

A Simple Example of C Code

```
/***************
* L. Fesq          *
* This function    *
* calculates the   *
* square of 5      *
****************/
int main (void)
{
      5 * 5;
      return 0;
}   /* end main */
```

Critiquing our Code

- This function computes the square of 5, but what does it do with the result?
- It would be helpful to store the result so we can use it again
- Use a variable to allocate memory to store the result, and to label it with a relevant name:

```
int main (void)
{
        int answer;
        answer = 5 * 5;
        return 0;
}
```

Critiquing our Code - cont.

- If we try running our simple example, what happens? Will we be able to see any results? How/why not?
- Add a line (or two) of code to help you see what is going on in this program.

```
#include <stdio.h>
int main (void)
{
        int answer;
        answer = 5 * 5;
        printf ("The square of 5 is %d\n", answer);
        return 0;
}
```

An Extension to our Simple Example

- The function would be more useful if it computes the square of any integer number
- How can we provide input to the program?

```
#include <stdio.h>
int main (void)
{
        int num;
        int answer;
        printf ("Enter the integer that you would like squared:  \n");
        scanf ("%d", &num);
        answer = num * num;
        printf ("The square of %d is %d\n", num, answer);
        return 0;
}
```

Another Extension to our Simple Example

- You can pass information from one function to another function when running the program
- An argument allows you to supply information to a function, such as the number that you want to square:

  <fcn-type> <fcn-name> (<argument-type argument>)

  1. How many arguments/inputs will be passed to the function? _____
  2. Argument type: What type of value will be passed? _____
  3. Argument name: What shall we call the argument? _____
- Create a new function *square*, and have *main* call *square* with an argument

Sample Code

```
#include <stdio.h>

int square ( int num)
{
        int answer;
        answer = num * num;
        printf ("The square of %d is %d\n", num, answer);
        return 0;
}

int main (void)
{
        square (5);
        return 0;
}
```

Sample Code for Returning a Value from a Function

```
#include <stdio.h>

int square ( int num)
{
    int answer;
    answer = num * num;
    printf ("The square of %d is %d\n", num, answer);
    return answer;
}

int main (void)
{
    int final;
    final = square (5);
    return 0;
}
```

Summary

- We now understand mechanics of program construction (post design)
  - Create source file
  - Type in C code
  - Build: compile and link
  - Execute: load and run
- Review structure of a C program - how do we write a C program?
  - Program consists functions, one of which must be *main*
  - Each function must follow Anatomy guidelines: fcn type, name, etc.
  - You get points for style - comments, indent, etc.
- For Monday
  - Read Chapter R4, especially 4.1 and 4.3. Skim other sections
  - Read Sections C2.1-C2.5