Introduction to C Programming

The computer will do exactly what you tell it to, rather than what you want it to do.

It will take at least twice as long to debug the code as it will to write the code.

C Overview

- C is a compiled language
  - Write code in one or more files
  - Compile each file
  - Link compiled files and library routines
- Must declare all variables (memory allocation)
- Variables have types (format)
- Many fewer built-in functions, but many libraries
- Pointers

Basic Program Structure

- Comments describing the program
- Header file inclusions (Libraries)
- Global definitions and macros
- Function and global variable declarations
- The main routine
- Function definitions

`helloworld.c`

Compiling and Linking

Before code is run it must be:

- Pre-processed (resolve `#include` and `#define`)
- Compiled (create object files)
- Linked (combine object files and libraries into an executable file)

Syntax: `gcc -o helloworld.exe helloworld.c`

- `gcc` is the C compiler
- `-o` specifies the output file
- `-O2` specifies optimization
- Libraries are linked with `-l`

Variable Types

- int
- float
- double
- array variants
- char, string (char[ ])
- pointer variants (`*`)
- more obsolete variants (long, short, unsigned, etc.)

`Cprog1.2.c`

`Cprog1.2b.c`

Math Notes

- $+ - * /$
- obeys standard order of operations
- assigns type (precision) from left to right
- %
- no built-in general exponentiation operator
- In `<math.h>`:
  - exp, log
  - round, ceil, floor
  - sin, cos, etc
  - and more

`Cprog1.3.c`
`Cprog1.3b.c`

Fun C operators

- `+=`
  - `i += 1` *means* `i = i + 1`
- `++`
  - `i++` *means* "evaluate this expression with the current value of i, and then make `i = i + 1`"
  - `++i` *means* "first make `i = i + 1` and then evaluate this expression"
  - `i--`

Loops

- `while`
  - `while(i<5)`
  - executed while the condition is true
  - also `do` ...`while`
- `for`
  - `for(i=0;i<5;i++)`
  - set starting value; execute while the condition is true; do the third step after each execution of the loop

`Cprog1.4.c`

Functions

- Declare function before `main`
- Define function after `main`
- Functions have the type of what they return
- Arguments must be listed *with types* in both declaration and definition
- Arguments are passed by value
- All variables are local

`Cprog1.5.c`

Pointers

- Each variable is stored somewhere in memory
- You can access these memory locations with pointers
- `&x` is the address of `x`
- `*p` is the value of the object stored at memory location `p`

`Cprog1.6.c`

Arrays

- Like a vector
  - Declaration: `double x[10];`
  - Accessing an element: `x[2];`
  - Note index is from 0 to $n-1$
- An array is actually a pointer
  - `x = &x[0]`
  - `x[2] = *(x+2)`
- Matrix `x[10][6]`
- Character string `char c[10];`

`Cprog1.7.c`

Conditional Statements

- if ... else if ... else
  - If the condition is true, it executes the immediately following statement and then skips to after the else.
  - Otherwise it checks each subsequent else if until it finds a true condition, or defaults to the else statement, assuming these exist (they are optional).
  - if clauses can be nested
- ternary operator `? :`
  - If condition to the left of `?` is true, return the expression to the left of the colon, otherwise return the expression to the right of the colon.
  - Example: `z = (a > b) ? a : b` *means* z is assigned the larger of a or b.

`Cprog1.8.c, bisect.c`

Error Messages

- Segmentation fault
  - There is some mistake in the code.
- Floating exception
  - An invalid numerical operation occurred (division by zero, square root of a negative number, overflow, etc.)
- Bus error
  - An error occurred with input/output (check to make sure that all specified file names are exactly right)
- core files
  - In .cshrc, add the line: `limit coredumpsize 0`
- killing runaway programs
  - Control-c
  - Get process number with `top` and use `kill`

Sorting and Recursion

- `qsort()` exists in `stdlib.h`
  - It uses the quicksort algorithm
  - It has some odd arguments, since it was written to be very general
- `quicksort.c` example
- Functions can be called recursively
- also in `quicksort.c` example

Random Numbers

- Built-in random integer generator `rand()`
- Do NOT use `rand()`
- Use `drand48()`, which is in the math library
- It is a good linear conguential random number generator, generating random U[0,1]
- Can set the seed with `srand48()`
- Of course, these are really pseudo-random numbers

Other Random Numbers

- U[a,b] from `(b-a)*drand48()+a`
- Uniform on integers from 1 to `k` with `ceil(drand48()*k)`
- Inverse CDF Transformation: To sample X from a distribution with CDF F, sample $U \sim U[0,1]$ and let $X = F^{-1}(U)$.
- Box–Muller transformation for normals
- other transformations (e.g., `-log(drand48())/`$\lambda \sim \mathrm{Exp}(\lambda)$)
- Library Code (e.g., Numerical Recipes)

Box–Muller Transformation

- Generates two independent N(0,1) from two independent U[0,1]
- Idea: let $X_1, X_2 \stackrel{iid}{\sim} N(0,1)$
- What is the distribution of $X_1^2 + X_2^2$?
- What is the distribution of $tan^{-1}\left(\frac{X_1}{X_2}\right)$?
- (It may help to think of the plot of X_1 vs. X_2.)

- $X_1^2 + X_2^2 \sim \chi_2^2 = Exp(\frac{1}{2})$
- $tan^{-1}\left(\frac{X_1}{X_2}\right) \sim U[0, 2\pi]$
- Thus let $U_1, U_2 \stackrel{iid}{\sim} U[0, 1]$
- Let $R = \sqrt{-2 * \log(U_1)},\ T = 2\pi * U_2$
- Let $X_1 = R * cos(T),\ X_2 = R * sin(T)$
- You can verify that this works by doing the transformation on the densities and computing the Jacobian.

`randnorm1.c, randnorm2.c`