Summary Page: Basic C
CISC 220, fall 2012

\

Sample Program:

File 1: `root.c`:

```
float absValue(float x) {
  if (x < 0)
    return -x;
  else
    return x;
} // end absValue

float squareRootGuess(float n, float guess) {
  return (guess + n / guess) / 2;
} // end squareRootGuess

float squareRoot(float n) {
  // Using Newton's method
  float guess = n/2;
  int done = 0; // false (not done yet)
  float newGuess;
  while (!done) {
    newGuess = squareRootGuess(n, guess);
    if (absValue(guess-newGuess) < .001)
      done = 1; // true
    else
      guess = newGuess;
  } // end while
  return newGuess;
} // end squareRoot
```

Commands to compile & link this program:

```
gcc -c root.c
gcc -c sqrt.c
gcc -o sqrt root.o sqrt.o
```

Executable program is now in `sqrt` file.

File 2: `root.h`:

```
float squareRoot(float n);
```

File 3: `sqrt.c`

```
#include <stdlib.h>
#include <stdio.h>
#include "root.h"

int main() {
  printf("enter a number: ");
  float num;
  int result = scanf("%f", &num);
  if (result != 1) {
    printf("Error: input is not a number\n");
    exit(1);
  }
  else {
    float root = squareRoot(num);
    printf("The square root of %f is %f\n", num, root);
    printf("%f squared is %f\n", root, root*root);
  } // end if

  return 0;
} // end main
```

Types:

`char`: a single character
`int`: an integer
`float`: a single-precision floating point number
`double`: a double-precision floating point number

There is no special string type: a string is simply an array of `char`s.
There is no boolean type: use `int`, with 0 meaning false and any non-zero value meaning true.

Pre-Processor:

```
#define SIZE 10
#if SIZE<20
                ....
     #else
        ....
#endif
```

other tests:

```
#ifdef SIZE
#ifndef SIZE
```

to remove a definition:

```
#undef SIZE
```

printf:

```
int printf(char *format, arg1, arg2, ....);
```

Conversion specifications which may be used in format string:

`%d`: integer
`%f`: floating-point number
`%c`: single character
`%s`: string

Minimum field width: a number directly after the "%" -- for example, `%8d`. If the output would not be 8 characters long, pads with spaces on the left (right justified). If the number is negative (`%-8d`), the output is left justified (spaces on the right).

`%8.2f`: minimum of 8 characters total, with *exactly* 2 digits after the decimal point
`%10.8s`: 8 = maximum length; extra characters cut off. Displayed using 10 characters -- so 2 spaces added on the left.

Result of `printf` is the number of items printed.

scanf:

```
int scanf(char *format, address1, address2, ....);
```

Format string contains `%d`, `%f`, etc. as for `printf`.

To read a value into a variable, put "&" before the variable name to get its address:

```
float f;
scanf("%f", &f);
```

Result of `scanf` is the number of items successfully read, or `EOF` if it hit the end of the input file before it could read anything.