CS 131

User Input

Command Line Arguments & `scanf`

Input from the command line

- Your program starts running at
  `int main(int argc, char* argv[])`
- Your program *is a function*, with two arguments!
  First argument: count of CLI words typed
  Second argument: collection of those CLI words

Program using CLI input

```c
#include <stdio.h>

int main(int argc, char* argv[])
{
    int index;
    printf("MAIN RUNS!\n");
    printf("argc is [%d]\n", argc);
    for(index=0; index<argc; index++)
    {
        printf("argv[%d] is %s\n", index, argv[index]);
    }
    printf("MAIN IS DONE!\n");
    return 0;
}
```

Another program with CLI inputs

```c
#include <stdio.h>
const int ARGUMENTS_EXPECTED=2;

int main(int argc, char* argv[])
{
    if(argc != ARGUMENTS_EXPECTED)
    {
        printf("ERROR: You must provide more information.\n");
        printf("USAGE: %s filename.c\n", argv[0]);
        return 1; //non-0, because something went wrong
    }

    printf("To compile, snapshot, and run your program, type:\n");
    printf("\tclang %s\n", argv[1]);
    printf("\tgit add %s\n", argv[1]);
    printf("\tgit commit -m \"your message here\"");
    printf("\t./a.out\n");

    return 0;
}
```

Input while the program is running ("runtime")

- New function: scanf

- A call to scanf...
  IS: an integer
  MEANS: the number of variables filled
  DOES: reads keyboard input, tries to match it to the 'format string' (first argument), and fills matched data into the variables whose addresses are given as 2nd through final arguments.

Example using scanf

```c
#include<stdio.h>

int main(int argc, char* argv[])
{
    int myvariable;

    printf("Enter a number: ");
    scanf("%d", &myvariable);
    printf("*You entered %d*", myvariable);

    return 0;
}
```

Syntax

- scanf(FORMAT_STRING, &var1, &var2, ...);
- FORMAT_STRING is like in printf, but describes the text *expected* (with blanks to ‘capture’) instead of the text to *produce* (with blanks to ‘fill’)
- var1, var2, etc are variable names to store data in -one name for each blank
- The &ampersand is **required** for variables in scanf

Input Stream

- Keyboard input is like a queue, waiting line, or dam-able stream: It builds up, and each `scanf` call only ‘consumes’ some of it.
- An attempt to ‘consume’ some of the stream when the stream is empty will **pause** the program until the user has added enough new data to.
- There is **no prompt** unless *you* supply one.

Invalid Input

- When scanf encounters input in the stream that does not match the format string, it matches (and ‘consumes’) as much as it can and then stops consuming.
- If you try to scanf for two data and only one (or none) get filled, scanf returns 1 (or 0) instead of 2