XSEDE Scholars Program
Introduction to C Programming

John Lockman III

June 14th, 2012

Introduction to C Programming

Outline

- Homework 3 review
- Recap Controls Constructs
  - If/then/else if
  - Switch Case
  - Loops
    - For
    - While
    - Do While
- I/O
  - Formatted I/O
    - printf
    - scanf
    - Other methods
  - File I/O
    - Reading files
    - Writing Files

Homework 3
Problem 1

- Write a program that prompts the user to enter two integers, adds the two integers, and then prints the sum of the integers to the screen.

Homework 3
Problem 1 – Solution

```c
#include <stdio.h>

int main(){

  int a, b, c;

  printf("Enter the first integer: ");
  scanf("%d", &a);
  printf("Enter the second integer: ");
  scanf("%d", &b);

  c = a + b;

  printf("Sum of two integers = %d\n", c);

  //or we can add the integers inside the printf function
  printf("Sum of two integers = %d\n", a+b);

  return 0;
}
```

Homework 3
Problem 2

- Write a C program that prompts the user to enter two integers, finds the larger of the two integers, and prints it to the screen.

Homework 3
Problem 2 - Solution

```c
#include <stdio.h>

int main(){

  int a, b;

  printf("Enter the first integer: ");
  scanf("%d", &a);
  printf("Enter the second integer: ");
  scanf("%d", &b);

  if(a > b)
    printf("%d is greater than %d\n", a, b);
  else
    printf("%d is greater than %d\n", b, a);


  return 0;
}
```

Recap – Control Constructs

If, Else, Else if

```
if (condition) {
    //if condition is true (!=0)
    // do something
}
else {
    //if condition if false (==0)
    // do something else
}
```

```
if (condition) {
    //if condition is true (!=0)
}
else if(2nd condition) {
    //if 2nd condition is true (!=0)
    //do something else
}
else {
   //if both above are false (==0)
}
```

Recap - Control Constructs

Switch, Case

```
switch(expression) {
   case const-expr: statement
   case const-expr: statement
   default: statement
}
```

Recap - Control Constructs

For, While and Do Loops

```
/* for loop */
for (initialiser ; condition ; increment) {
    code;
}
```

```
/* do loop */
initialiser;
do {
    code;
    increment;
} while (condition);
```

```
/* while loop */
initialiser;
while (condition) {
    code;
    increment;
}
```

Formatted I/O

Formatted output (control characters)

```
main()
{
   printf("Hello, ");
   printf("world");
   printf("\n");
}
```

\a	**bell**
\b	**backspace**
\n	**newline**
\r	**carriage return**
\t	**horizontal tab**
\v	**vertical tab**
\f	**formfeed**
****	**backslash**
\'	**single quote**
\"	**double quote**

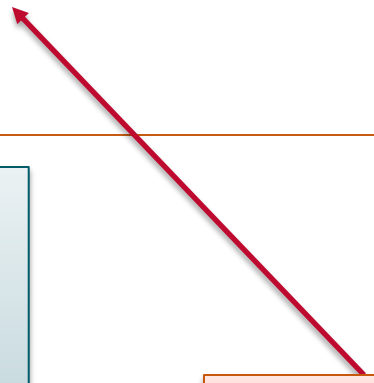
Control character: newline

Formatted I/O

Formatted output (conversion characters)

```
#include <stdio.h>
int main()
{
    int location;
    float rain_fall;
    …
    printf("City code %d ", location);
    printf("has rainfall %f", rain_fall);
    printf("\n");
    return 0;
}
```

%d	print decimal integer
%f	print floating point
%c	print character
%s	print string of characters
%6d	print decimal at least 6 chars wide
%.2f	print float until 2 chars after point



Conversion character

Formatted I/O

More printf() examples

```
#include <stdio.h>
int main(){
   char myName[20] = "Fred";
   int myAge = 42;
   printf("My name is %s and I am %d yrs old\n", myName, myAge);
   return 0;
}
```

Carrige control is provided by the backslash "\"

- \n = newline
- \t = tab
- \b = backspace
- \\ = produce a single backslash "\"

Formatted I/O

printf(), putting it all together

- printf() is a highly robust function that can parse output effectively.

```
printf("\n\nHOST\t\tR15s\tR1m\tR15m\tPAGES\t\tMEM\tSWAP\tTEMP\n");

printf("%s\t%5.1f\t%5.1f\t%5.1f\t%5.1fP/s\t%4.fM\t%4.fM\t4.fM\n",
hosts[i].hostName,hosts[i].li[R15S],hosts[i].li[R1M],hosts[i].li[R1
5M],hosts[i].li[PG],hosts[i].li[MEM],hosts[i].li[SWP],hosts[i].li[T
MP]);
```

Output:

```
HOST    R15s    R1m     R15m    PAGES       MEM    SWAP    TEMP
c21-208 1.0     1.0     1.0     2.1P/s      6468M  2021M   55776M
```

Formatted I/O

Formatted input

- Input can be handled with the scanf() function which behaves similarly to printf()

```
int number, check;
float myFloat;

check = scanf ("%d %f",&number, &myFloat);
if (check != 1) {
    printf ("Error!\n");
    return -1;
}
```

Conversion character

Why &?

Formatted I/O

Formatted input

- Variables must be handed to scanf() with the address operator “&”. In C the address operator gives a variable’s address in memory (a pointer), not the value itself.

```
int number, check;
float myFloat;

check = scanf ("%d %f",&number, &myFloat);
if (check != 1) {
    printf ("Error!\n");
    return -1;
}
```

Formatted I/O

Working with scanf()

- scanf()
  - will parse standard input and match cases that it finds.
- gets(string)
  - Grabs all of the string input on the screen and will save it in the array string
  - It is NOT recommended to use gets (why?)

Formatted I/O

Why Not gets()?

- According to the MAN page for gets():

BUGS

Never use **`gets()`** because it is impossible to tell without knowing the data in advance how many characters **`gets()`** will read, and because **`gets()`** will continue to store characters past the end of the buffer, it is extremely dangerous to use. It has been used to break computer security. Use **`fgets()`** instead.

Formatted I/O

Working with Files

New header file: string.h

strcpy()

```
/* definitions */
#include <string.h>
#define NAMELIMIT 40
FILE *fptr;
char filename[NAMELIMIT];
strcpy(filename,"foo.dat");

/* in code */
fptr = fopen(filename, "r");
if ( fptr == NULL) {
    perror("error opening file");
}
fclose (fptr);
```

Formatted I/O

Working with Files

- The library function `fopen()` opens a file and returns a predefined structure: `File *`
- NULL is returned on error

```
FILE *fptr;
char filename[] = “foo.dat”;

fptr = fopen(filename, "r");
if ( fptr == NULL) {
    perror("error opening file");
}
fclose (fptr);
```

Formatted I/O

File Modes

```
FILE *fopen(const char *filename, const char *mode)
```

- The second argument of `fopen` is the *mode* in which we open the file. There are three
  - "r" opens a file for read only
  - "w" opens a file for write only – writing over any previous content (i.e. deletes the content of the file)
  - "a" opens a file for appending - writing at the end of the file

Formatted I/O

File Modes

```
FILE *fptr;
fptr = fopen( "myfile","r");
//Open file for reading
```

Mode	Definition
r	Reading
w	Writing
a	Append write
r+	Read & Write
w+	Read & Overwrite
a+	Append read/write

Formatted I/O

File Pointers

- File pointers can be manipulated in the same fashion as I/O from the keyboard and to the screen.
- `fprint()` `fscanf()` can read and write to a file pointer.

```
int item
fscanf (fptr, "%d", &item); /* read one value */
printf( "%d\n", item); /* print item to screen */
```

Formatted I/O

fprintf()

- `fprintf()` is just like `printf()` except that it writes to a file referred to in a file pointer.
- `fscanf()` is also like `scanf()`, except that it reads from a file referred to in a file pointer.

```
int x;
FILE *fptr;
fptr= fopen ("file.dat",“rwa");

fscanf (fptr,“%d“, &x);

fprintf (fptr,"Hello World!\n");
```

Formatted I/O

fgets()

- fgets may be a better way to read from a file
- The function reads a line into a string

```
FILE *fptr;
char line [1000];
/* Open file and check it is open */
while (fgets(line,1000,fptr) != NULL) {
   printf ("Read line %s\n",line);
}
```

•fgets takes 3 arguments, a string, a maximum number of characters to read and a file pointer. It returns NULL if there is an error (such as EOF)

Formatted I/O

Closing a file

- fclose() flushes its write buffer and closes the file.

```
FILE *fptr;
char filename[]= "myfile.dat";
fptr= fopen (filename,"w");
if (fptr == NULL) {
    printf ("Cannot open file to write!\n");
    exit(-1);
}
fprintf (fptr,"Hello World of filing!\n");
fclose (fptr);
```

Formatted I/O

exit() and abort()

- On encountering an error we may want to “immediately exit" from program execution.
- In functions we can use `exit()` or `abort()` to do this.
- `exit()` and `abort()` are part of the stdlib.h library
- `abort()` generates a “core” file which can be used for debugging.

Formatted I/O

Common C Mistakes

- Don’t use the name of the file to close it

```
FILE *fptr;
fptr= fopen ("myfile.dat","r");
/* Read from file */
fclose ("myfile.dat");
/* That's wrong */
```

```
FILE *fptr;
fptr= fopen ("myfile.dat","r");
/* Read from file */
fclose (fptr);  //correct method to close
```

Formatted I/O

Common C mistakes

- fgets reads and includes the '\n' at the end!
- This can be a problem - for example if we got input from the user and tried to use it to write a file:

```
FILE *fptr;
char readfname[1000];
fgets (readfname,1000,stdin);
fptr= fopen (readfname,"w");
/* this will return an error since
   file name has \n */
```

Formatted I/O

Three Special Streams

- Three special file pointers are defined in the `stdio.h` header
  - `stdin` refers to standard input from the keyboard
  - `stdout` refers to standard output on the screen
  - `stderr` refers to the standard error device (usually also the screen).
- This is the same as printf():

```
fprintf (stdout,"Hello World!\n");
```

Formatted I/O

Common File Reading

- It is quite common to want to read every line in a program.

```
/* define MAXLEN at start using enum */
FILE *fptr;
char tline[MAXLEN];  /* A line of text */
fptr= fopen ("sillyfile.txt","r");
/* check it's open */
while (fgets (tline, MAXLEN, fptr) != NULL) {
    printf ("%s",tline); // Print it
}
fclose (fptr);
```

Formatted I/O

Common Keyboard Reading

- `fgets()` and `stdin` can be combined to get a safe way to get a line of input from the user

```
#include <stdio.h>
int main()
{
    const int MAXLEN=1000;
    char readline[MAXLEN];
    fgets (readline,MAXLEN,stdin);
    printf ("You typed %s",readline);
    return 0;
}
```

Formatted I/O

Converting Strings to Numbers - `atoi()` & `atof()`

- we can use `atoi` or `atof` to convert a string to a number
- The functions are part of `stdlib.h`

```
char numberstring[]= "3.14";
int i;
double pi;
pi= atof (numberstring);
i= atoi("12");
```

Homework 4
Basic Math Problems

1. Basic Math Operations
   - Print a welcome message to the screen explaining what the program does.
   - Prompt the user for two floating point numbers
   - Prompt the user for an operation: +, - , *, or /
   - Calculate the result of the two operators and the operand using a Switch Case statement
   - Print the result to the screen and quit the program
2. Summing Integers
   - Print a welcome message to the screen explaining what this program will do.
   - Prompt the user for two integers, a lower bound and an upper bound
   - Sum up all the integers between the lower bound and the upper bound, inclusive, using a **for** loop".
   - Print the sum to the screen and quit the program