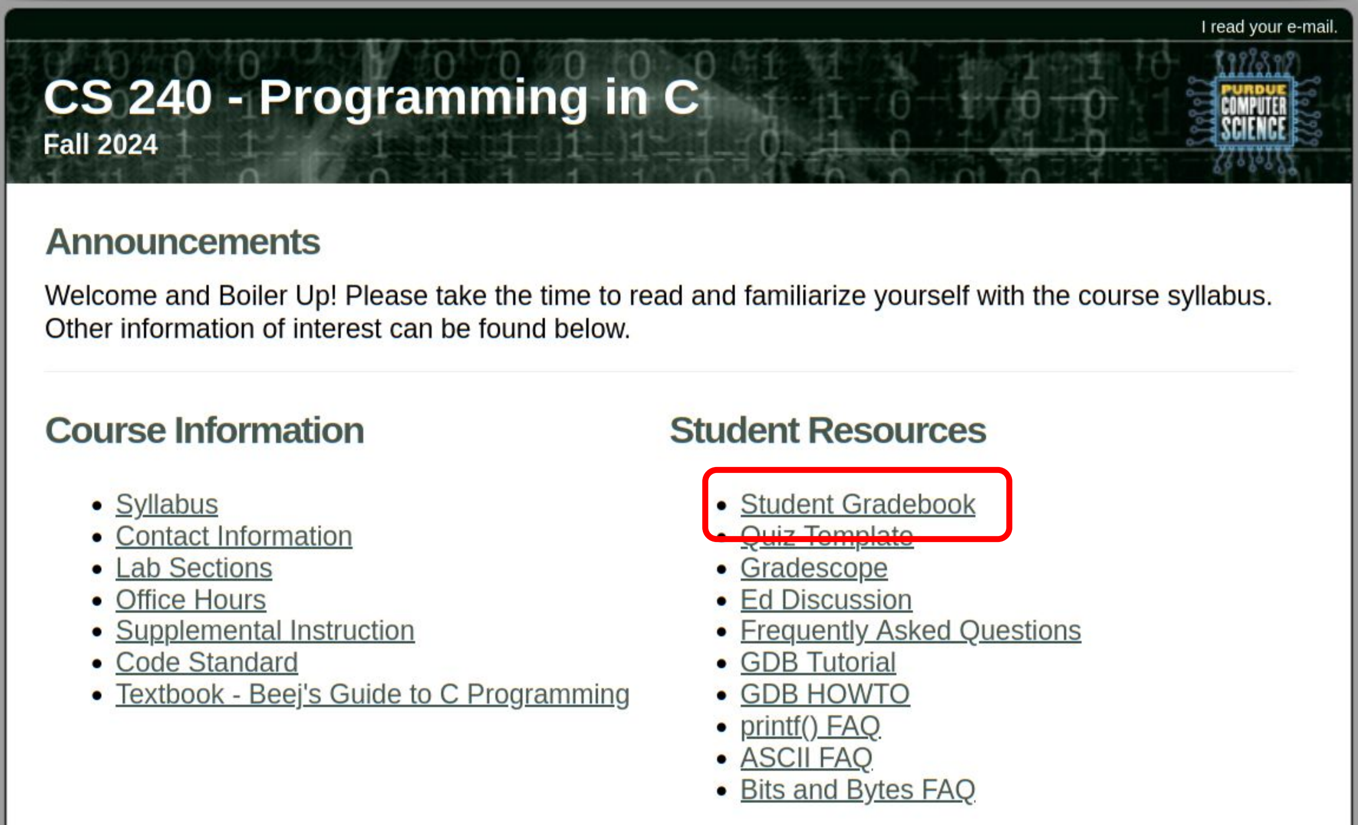
CS 240: Programming in C

Lecture 4: More File I/O

Announcements

- Homework 0 grades are released

Gradebook



I read your e-mail.

CS 240 - Programming in C

Fall 2024

Announcements

Welcome and Boiler Up! Please take the time to read and familiarize yourself with the course syllabus. Other information of interest can be found below.

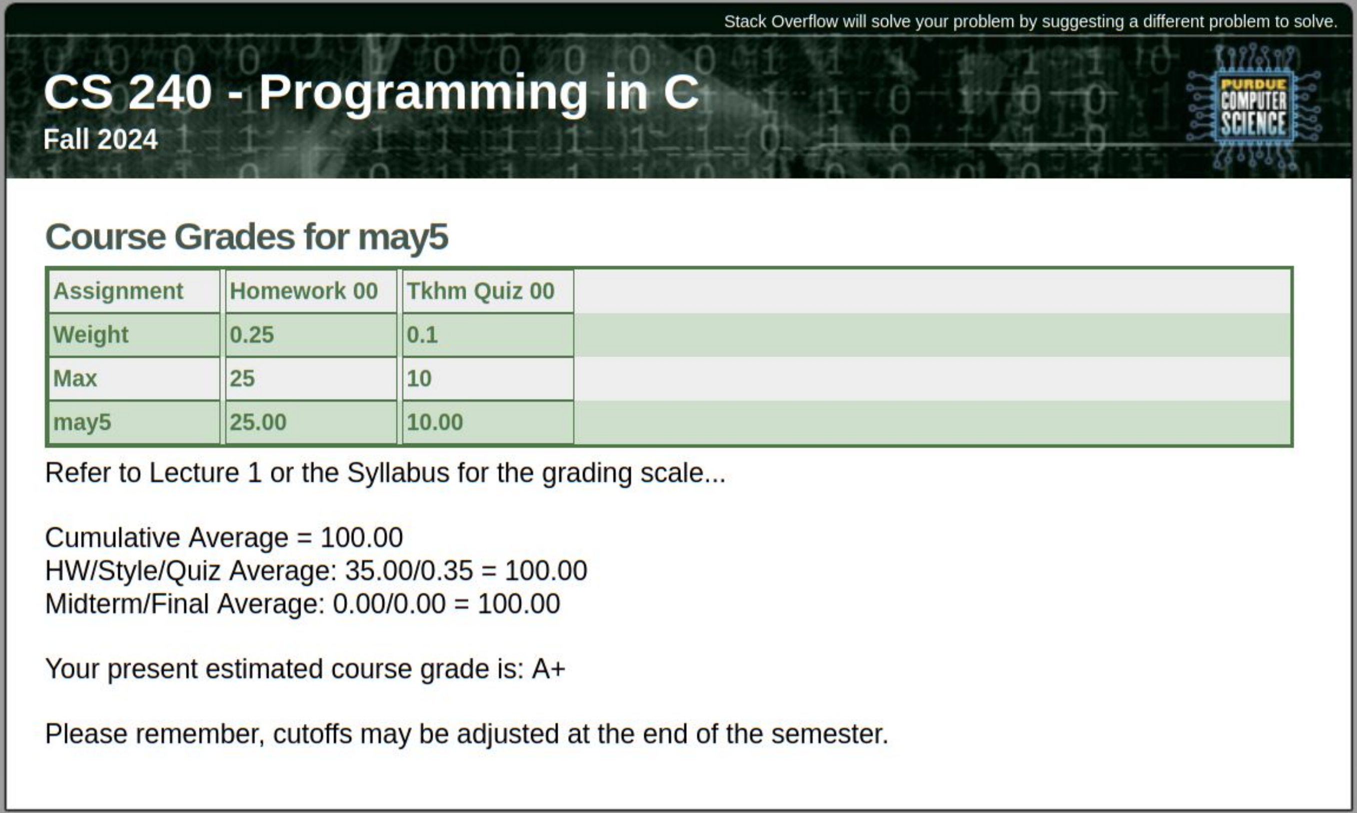
Course Information

- Syllabus
- Contact Information
- Lab Sections
- Office Hours
- Supplemental Instruction
- Code Standard
- Textbook - Beej's Guide to C Programming

Student Resources

- Student Gradebook
- Quiz Template
- Gradescope
- Ed Discussion
- Frequently Asked Questions
- GDB Tutorial
- GDB HOWTO
- printf() FAQ
- ASCII FAQ
- Bits and Bytes FAQ

Gradebook



Stack Overflow will solve your problem by suggesting a different problem to solve.

CS 240 - Programming in C

Fall 2024

Course Grades for may5

Assignment	Homework 00	Tkhm Quiz 00
Weight	0.25	0.1
Max	25	10
may5	25.00	10.00

Refer to Lecture 1 or the Syllabus for the grading scale...

Cumulative Average = 100.00
HW/Style/Quiz Average: 35.00/0.35 = 100.00
Midterm/Final Average: 0.00/0.00 = 100.00

Your present estimated course grade is: A+

Please remember, cutoffs may be adjusted at the end of the semester.

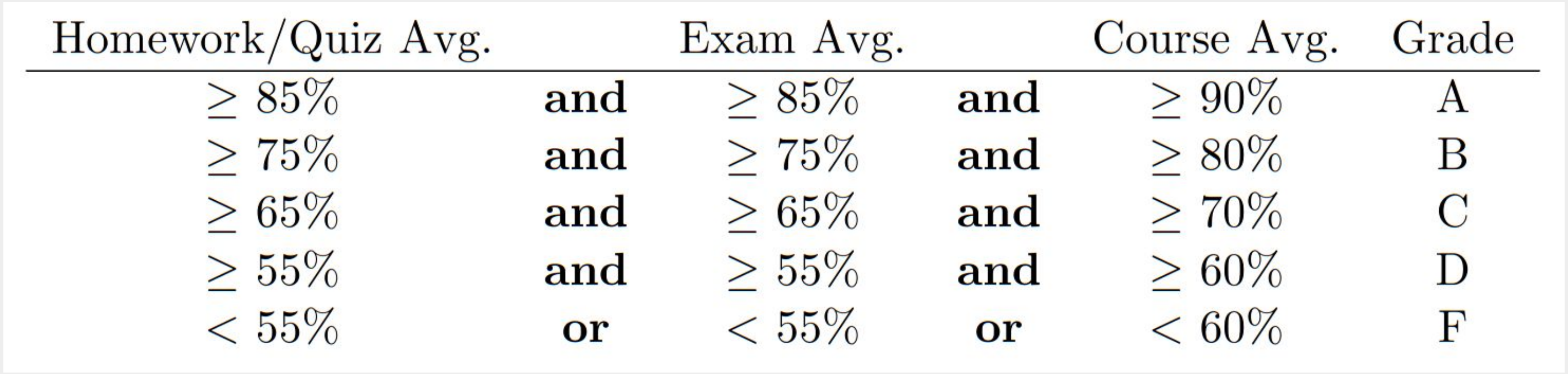
Grade Calculation

- HW/Style/Quiz (HSQ) and Exams (separately):
  - Σ_i (score[i] / max[i] * weight[i]) * 100
- Cumulative Average:
  - (HSQ avg * 0.5) + (Exam avg * 0.5)

PURDUE UNIVERSITY®

Grade Cutoffs

- Limited by lowest category (HSQ or Exams)
- Both within 5% of cumulative Avg for letter grade
    - Applies to +/- as well
    - e.g., A+: 97% cumulative avg *and* minimum 92% HSQ & Exam
- Exact cutoffs subject to change



Homework/Quiz Avg.		Exam Avg.		Course Avg.	Grade
$\geq 85\%$	**and**	$\geq 85\%$	**and**	$\geq 90\%$	A
$\geq 75\%$	**and**	$\geq 75\%$	**and**	$\geq 80\%$	B
$\geq 65\%$	**and**	$\geq 65\%$	**and**	$\geq 70\%$	C
$\geq 55\%$	**and**	$\geq 55\%$	**and**	$\geq 60\%$	D
$< 55\%$	**or**	$< 55\%$	**or**	$< 60\%$	F

Reading data in C

- C is a little different than Java when it comes to reading data
- Think `printf()` in reverse…

```
scanf(“%s %d”, buffer, &int_var);
```

- Returns the number of successful conversions

scanf()

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

- Reads characters from stdin, interprets them according to `format`
- Stores results in the **locations** given by the following arguments
- `format` string determines the number and type of following arguments

scanf() example

```
#include <stdio.h>

int main() {
   int x;
   scanf(“%d”, &x);

   printf(“%d\n”, x);
   return 0;
}
```

scanf() example

```
#include <stdio.h>

int main() {
   int x;
   char y[20];
   scanf(“%d %s”, &x, y);

   printf(“%d %s\n”, x, y);
   return 0;
}
```

PURDUE UNIVERSITY

Format string

- Whitespace character
    - Read and skip any whitespace in the input stream
- Non-whitespace character (except %)
    - Read next character in the input stream, compare to this character
    - If it matches, discard and continue, otherwise abort
- Format specifiers
    - Starts with %
    - d, f, s, c, etc.
    - Can also specify field width: the max number of chars to read

Format string

```
scanf("%d, %d, %d, %d", &n1, &n2, &n3, &n4);
```

- Read four integers separated by commas
- Which lines match?

```
✓ 1, 2, 3, 4
✓   10,   4,72,\t   65535
✗ 99,98,97
✗  12 ,34 ,56 ,78
✓ 9, 8, 7, 6, 5, 4, 3, 2, 1
```

%[] - set of characters

- Specify a set of allowed characters
  - %[ACGT] matches any sequence of A, C, G, and T characters
- Can also use ranges
  - %[0-9], %[A-Z], %[0-9A-Za-z]
- Can invert
  - %[^0-9], %[^\n]

- For %[] and %s, **field width** is important

scanf() example

```
char buffer[100];
int val;
float cash;
int x;
x = scanf(“%s, %d, %f”, buffer, &val, &cash);
```

- Is it correct?
- What will a matching string look like?

PURDUE UNIVERSITY

scanf() example

```
char buffer[100];
int val;
float cash;
int x;
x = scanf(“%s, %d, %f”, buffer, &val, &cash);
```

- Is it correct?
- What will a matching string look like?
  - You might think: `abc, 123, 3.14`
  - But %s matches until it reaches a whitespace char

scanf() example

```
char buffer[100];
int val;
float cash;
int x;
x = scanf("%99[^,], %d, %f", buffer, &val, &cash);
```

- `%99[^,]` reads up to 99 characters, or until it finds a comma
- `abc, 123, 3.14` will match correctly

PURDUE UNIVERSITY

Try it on your own!

- Try to implement that previous example without looking at the slide first
- Write it down by yourself
- Type it in by yourself
- Modify it to read different patterns
- Lookup different format specifiers and test them out
- Practice will improve your skills and understanding

Return value

- Returns the number of variables successfully scanned
- Or EOF if we’ve hit the end of the file
- It may return zero if there’s a blank line at the end of the file
- To find the end of the file, check if the return value is either 0 or EOF
- To catch any other error, check if the return value is less than the number of variables to scan

More file operations

```
int access(char *file_name, int mode);
```

- Used to check that a file can be opened with the specified mode

```
int feof(FILE *file_pointer);
```

- Used to determine if end-of-file was reached by the **previous** read

```
int ferror(FILE *file_pointer);
```

- Check for any error condition for the file

```
void clearerr(FILE *file_pointer);
```

PURDUE UNIVERSITY

access()

- Used to check if a file can be accessed before trying to open it

```
int access(char *file_name, int mode);
```

- Mode is not the same as fopen()'s mode. It's one of:
  - **`R_OK`**: Check for read access
  - **`W_OK`**: Check for write access
  - **`F_OK`**: Check for existence
- Returns zero on success

PURDUE UNIVERSITY

Example of access()

```
#include <stdio.h>
#include <unistd.h>  /* for access */

int main() {
    char file_name[100];
    int ret = scanf(“%99s”, file_name);
    if (ret != 1) {
        printf(“Specify a file.\n”);
        return 1;  /* indicate user error to OS */
    }

    if (access(file_name, R_OK) == 0) {
        printf(“%s is readable.\n”, file_name);
    } else {
        printf(“%s is not readable.\n”, file_name);
    }
    return 0;  /* indicate success to OS */
}
```

feof()

```
int feof(FILE *file_pointer);
```

- When we're reading from a file, we need to be able to determine when we've hit the end!
- Indicates EOF (end-of-file) **after** the last valid read
- Returns non-zero if we have hit EOF

A program that uses feof()

```
#include <stdio.h>

int main() {
     char file_name[100];
     scanf(“%s”, file_name);  /* skip error checking for brevity */
     FILE *fp = NULL;
     char buf[100] = “”;
     fp = fopen(file_name, “r”);

     fscanf(fp, “%[^\n]\n”, buf);      /* read a line  */
     while (feof(fp) == 0) {           /* test for EOF */
         printf(“%s\n”, buf);          /* print line   */
         fscanf(fp, “%[^\n]\n”, buf);  /* read a line  */
     }
     fclose(fp);
     return 0;
}
```

A program that doesn’t use feof()

```
#include <stdio.h>

int main() {
     char file_name[100];
     scanf(“%s”, file_name);  /* skip error checking for brevity */
     FILE *fp = NULL;
     char buf[100] = “”;
     fp = fopen(file_name, “r”);

     int status = 0;
     while (1) {
         status = fscanf(fp, “%[^\n]\n”, buf);
         if (status == 1) printf(“%s\n”, buf);
         else break;
     }
     fclose(fp);
     return 0;
}
```

Other errors…

- What if you run out of disk space while you’re writing a file? How do you check for this?
- ferror() looks for various errors (including EOF)
- Call it after every read or write to figure out if something bad happened
- If you can correct it somehow, use clearerr() to clear the file pointer’s internal error flag

ferror() example

```
#include <stdio.h>

int main() {
     char file_name[100];
     scanf(“%s”, file_name);  /* skip error checking for brevity */
     FILE *fp = NULL;
     fp = fopen(file_name, “w”);

     do {
         fprintf(fp, “Hello, world.\n”);
     } while (ferror(fp) == 0);

     printf(“The disk is full!\n”);
     return 0;
}
```

For next lecture

- Keep working on HW1
- (Re-)Read Chapter 7 of K&R
    - and/or Chapter 13 in Beej
- Understand the following functions:
    - ftell()
    - fseek()
    - fgets()
    - fputs()
    - assert()

Slides

- Slides are heavily based on Prof. Turkstra's material from previous semesters.