**University of Washington – Computer Science & Engineering**
Spring 2026 Instructor: Naomi Alterman 2026-05-04

# CSE 333 MIDTERM

| | | |
|---|---|---|
| Last Name: | | |
| First Name: | | |
| Student ID Number: | | |
| Name of person to your Left \| Right | | |
| All work is my own. I had no prior knowledge of the exam content, nor will I share the content with others in CSE 333 who haven't taken it yet. Violation of these terms could result in a failing grade. **(please sign)** | | |

## Do not turn the page until you are told to do so.

## Instructions

- This quiz contains 14 pages, including this cover page. Show scratch work for partial credit, but put your final answers in the boxes and blanks provided.
- The last sheet of paper is a reference sheet. *Detach it from the rest of the exam when you start.*
- The exam is closed book. You are allowed one page (US letter, double-sided) of *handwritten* notes.
- Please silence and put away all cell phones and other mobile or noise-making devices. Remove all hats, headphones, smart glasses, watches, and other digital wearables.
- You have 50 minutes to complete this exam.

## Advice

- Read questions carefully before starting. Skip questions that are taking a long time.
- Read *all* questions first and start where you feel the most confident.
- Relax. If you've been practicing, you got this. If you haven't, you'll learn something now.

| Question | 1 | 2 | 3 | 4 | 5 | Total |
|---|---|---|---|---|---|---|
| Possible Points | 10 | 18 | 15 | 20 | 1 | **64** |

(this page left blank for scratch space)

Student ID: _____.

# 1. Makin' my way downtown (makefiles, 10 pt)

Our friend Jimin is working on an application suite related to choreographing dances. Here's a preview of the source code:

| Moves.h | Steps.h | Steps.cc |
|---|---|---|
| `struct Wiggle {...};`<br>`struct Shimmy {...};` | `class Steps {...};` | `#include "Steps.h"`<br>`#include "Moves.h"` |

| Choreograph.cc | DanceInfo.cc |
|---|---|
| `#include "Moves.h"`<br><br>`int main(...) {...}` | `#include "Steps.h"`<br><br>`int main(...) {...}` |

The Makefile for this project has targets to build (or rebuild) the **`Choreograph`** and **`DanceInfo`** executables, as well as a default target named **`all`** that builds both. When we run **make all** we end up with the two programs and the object file **Steps.o**. No other files are generated in the process. Draw the dependency diagram for the Makefile, with arrows that point from targets to the predecessors they depend upon. [10 pt]

# 2. Keep it Classy (C++ Classes, 18pt)

Consider the following C++ program which does compile and execute successfully.

```
#include <iostream>
using namespace std;

class bar {
 public:
  bar() : num_(new int())       { cout << "j"; }
  bar(int a) : num_(new int(a)) { cout << "y"; }
  ~bar()                        { cout << "l"; }

 private:
  int* num_;
};

class baz {
 public:
  baz()                                 { cout << "d"; }
  baz(int a) : morebar_(new bar(a))     { cout << "u"; }
  baz(int a, int b): val_(b)            { cout << "o"; }

 private:
  bar bar_;
  bar* morebar_;
  int val_;
};

class foo {
 public:
  foo()                       { cout << "e"; }
  foo(int a, int b, int c)
      : baz_(a, c), bar_(b)   { cout << "f"; }
  ~foo()                      { cout << "u"; }

 private:
  baz baz_;
  bar bar_;
};

int main() {
  foo f(7, 9, 4);
  return EXIT_SUCCESS;
}
```

(a) Write the output that is produced when the program is executed. [10 pt]

(b) Unfortunately, the code above is leakin' memory all over the place! How many bytes of memory will be leaked when `main()` exits? Describe a way to fix this problem. [3 pt]

(c) Suppose we add the following public methods to the class baz:

```
int baz::GetValue() { return val_; }

bool baz::Compare(baz &other_baz) const {
   return val_ > other_baz.GetValue();
}
```

And then try to compile this function:

```
void wump() {
      const baz b1(1, 2);
      baz b2(2, 3);
      cout << b2.GetValue();      // 1
      b1.Compare(b2);             // 2
      b2.Compare(b1);             // 3
}
```

Which, if any, of the three numbered lines in `wump()` could cause a compiler error? Why? [5 pt]

# 3. A lot to process (preprocessor, 15pt)

Suppose we have the following three .h files and one .c file. On the next page, show the output produced by the C preprocessor when it processes file transaction.c with the following command:

```
cpp -E -P -DCOUPON=0.5 transaction.c
```

**Remember!** The C preprocessor only does string substitution and does not analyze the code it produces for correctness.

| utils.h | costs.h |
|---|---|
| `int getQuantity(int id);` | `#include "utils.h"`<br><br>`typedef float f;`<br><br>`f getPrice(int id);` |
| **users.h** | **transaction.c** |
| `#include "utils.h"`<br><br>`#define ITEM 1000`<br><br>`#define USER 2`<br><br>`void add_user(int id);` | `#include "costs.h"`<br><br>`#ifndef COUPON`<br>`#define COUPON 1.0`<br>`#endif`<br><br>`#define PRICE(x) (x + 5.0)`<br>`#define ITEM_BASE 300`<br>`#define ITEM ITEM_BASE + 33`<br><br>`int main(int argc, char** argv) {`<br>`  int Q = getQuantity(ITEM);`<br>`  f cost = Q * getPrice(ITEM);`<br>`  printf("cost: %f", PRICE(cost) * COUPON);`<br>`}` |

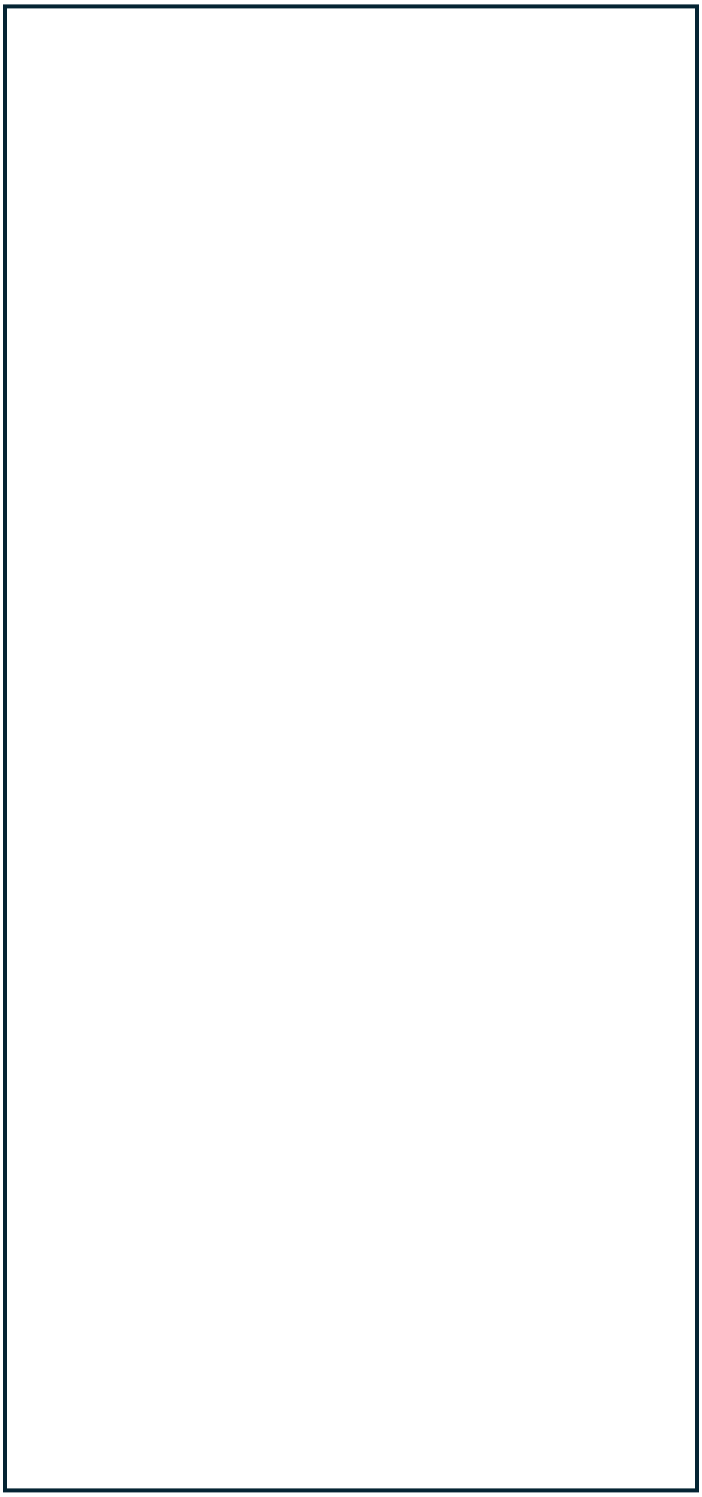
Result of `cpp -E -P -DCOUPON=0.5 transaction.c` :

# 4. Stacks are like onions; they have layers (memory diagrams, 20 pt)

Consider the following C++ program which compiles and runs successfully:

```
#include <iostream>

using std::cout;
using std::endl;

class Student {
   public:
   Student(char initial, int year)
       : initial_(initial), year_(year) { }

   int GetYear() const { return this->year_; }
   void SetYear(int y) { this->year_ = y; }
   int GetInitial() const { return this->initial_; }

   private:
   char initial_;
   int year_;
};

bool operator<(Student lhs, Student rhs) {
   int lhs_year = lhs.GetYear();
   int rhs_year = rhs.GetYear();
   // PART A
   return lhs_year < rhs_year;
}

int main() {
  Student farquaad('F', 5);
  Student shrek('S', 1);

  // PART B

  if (farquaad < shrek) {
    cout << "Farquaad is in a lower grade than Shrek" << endl;
  } else {
    cout << "Farquaad is NOT in a lower grade than Shrek" << endl;
  }

  return EXIT_SUCCESS;
}
```

(a) Draw a block-and-arrow diagram below to represent the stack of the process when it reaches the line labelled “PART A”. Clearly display the names, divisions between, and order of the stack frames. Assume high addresses are at the top of the diagram.
[15 pt]



(b) Suppose on the line indicated with “PART B”, we add these lines of code to the program:

```
Student& shrek_twin = shrek;
shrek_twin.SetYear(2);
```

**Briefly** explain what would change in the memory diagram in Part A and why. [5 pt]

## 5. Call me crazy (syscalls, 1 pt)

Make up your own syscall and tell us what it does. Any answer gets a point [1 pt]

(this page left blank for scratch space)

# CSE 333 Reference Sheet (Midterm)

## C Library Header – `stdio.h`

```
FILE          // type of object containing info to control a stream

FILE*  fopen (const char* filename, const char* mode);
int    fclose (FILE* stream);
int    fprintf (FILE* stream, const char* format, ...);
char*  fgets (char* str, int num, FILE* stream);
size_t fread (void* ptr, size_t size, size_t count, FILE* stream);
size_t fwrite (const void* ptr, size_t size, size_t count, FILE* stream);
void   perror (const char* str);
int    ferror (FILE* stream);       // returns non-zero if error on stream
```

## C Library Header – `stdlib.h`

```
EXIT_SUCCESS  // success termination code
EXIT_FAILURE  // failure termination code

void*  malloc (size_t size);
void*  realloc (void* ptr, size_t size);  // change size of mem block *ptr
void   free (void* ptr);                  // does nothing when ptr = NULL
void   exit (int status);                 // terminate calling process
```

## C Library Header – `string.h`

```
size_t strlen (const char* str);           // # of chars, not including '\0'

char*  strcpy (char* dst, const char* src);          // copy chars
char*  strcat (char* dst, const char* src);          // append chars
int    strcmp (const char* str1, const char* str2);  // compare strings
```

- Versions that take a third parameter `size_t num`: `strncpy()`, `strncat()`, `strncmp()`

## C Library Header – `math.h`

```
INFINITY      // Infinity
NAN           // Not-A-Number

float  abs (float x);                // absolute value
float  pow (float base, float exp);  // base raised to the power exp
float  sqrt (float x);               // square root
float  ceil (float x);               // round up (towards +∞)
float  floor (float x);              // round down (towards -∞)
```

- All of these functions are overloaded to work with `double`, too

## POSIX Library Headers – `fcntl.h`, `unistd.h`, `dirent.h`

```
O_RDONLY      // read-only flag
O_WRONLY      // write-only flag
O_RDWR        // read-write flag
O_APPEND      // append (add to end) flag
DIR           // type representing a directory stream

int     open (char* pathname, int flags, ...);          // open a file
int     close (int fd);                                 // close a file
ssize_t read (int fd, void* buf, size_t count);         // read from file
ssize_t write (int fd, const void* buf, size_t count);  // write to file

DIR*    opendir (const char* dirname);                  // open a directory
int     closedir (DIR* dirp);                           // close a directory
struct dirent* readdir (DIR* dirp);                     // read a directory
```

## Error Library – `errno.h`

```
errno         // # of the last error, usually checked against defined consts

EACCES        // permission denied
EBADF         // bad file/directory descriptor
EFAULT        // bad address supplied
EINTR         // interrupted function
EISDIR        // is a directory
ENOTDIR       // is not a directory
```

## C++ Memory Allocation

```
new           // allocate space for type, return pointer
new[]         // allocate space for array of type, return pointer
delete        // deallocate space indicated by pointer
delete[]      // deallocate space of array indicated by pointer
```

## Format Specifiers

| Specifier | Type | Specifier | Type |
|---|---|---|---|
| d / i | signed decimal int | f | decimal float |
| u | unsigned decimal int | c / s | character / string |
| x | unsigned hex int | p | pointer address |

## Streams

| <stdio.h> | POSIX | <iostream> |
|---|---|---|
| stdin | 0 | std::cin |
| stdout | 1 | std::cout |
| stderr | 2 | std::cerr |

## Makefile Automatic Variables

| | | |
|---|---|---|
| `$@  # target name` | `$^  # all sources` | `$<  # left-most source` |