Do not turn the page until 11:30.

Instructions

- This exam contains 8 pages, including this cover page. Show scratch work for partial credit, but put your final answers in the boxes and blanks provided.
- The exam is closed book (no laptops, tablets, wearable devices, or calculators). 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.
- 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. You are here to learn.

<table>
<thead>
<tr>
<th>Question</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>Total</th>
</tr>
</thead>
<tbody>
<tr>
<td>Possible Points</td>
<td>13</td>
<td>8</td>
<td>12</td>
<td>6</td>
<td>28</td>
<td>19</td>
<td>25</td>
<td>1</td>
<td>112</td>
</tr>
</tbody>
</table>
Question 1:

Consider the dependency graph, below, which was derived from our project’s Makefile.

(A) [4 pts] If DocTable.h is modified, which targets need to be rebuilt?

**DocTable.o, searchshell.o, searchshell**

(B) [4 pts] If DocTable.c is modified, which targets need to be rebuilt?

**DocTable.o, searchshell**

(C) [4 pts] In HW2, MemIndex.c contained a line to `#include "DocTable.h"`. The Makefile snippet which generated our dependency graph is below. What, if anything, needs to change in it?

X Changes Are Required to Makefile (see below)  □ No Changes Necessary

```
MemIndex.o: MemIndex.c MemIndex.h HashTable.h DocTable.h
  $(CC) $(CFLAGS) -c $<
```

(D) [3 pts] If changes are necessary to the Makefile, please describe how these changes would impact your answers to (A) and (B).

X Changes Are Required to (A) and (B) (described below)  □ No Changes Necessary

Part (A) needs to add MemIndex.o
Question 2:

[8 pts] Of the following, which are POSIX system calls and which are not?

<table>
<thead>
<tr>
<th>Syscall</th>
<th>Not Syscall</th>
</tr>
</thead>
<tbody>
<tr>
<td>struct dirent* readdir(DIR *dirp);</td>
<td>x</td>
</tr>
<tr>
<td>int open(const char *pathname, int flags);</td>
<td>x</td>
</tr>
<tr>
<td>void exit(int status);</td>
<td>x</td>
</tr>
<tr>
<td>size_t fread(void *ptr, size_t size, size_t count, FILE *stream);</td>
<td>x</td>
</tr>
</tbody>
</table>

Question 3:

[12 pts] Recall that the steps of creating and running a program are: preprocessing, compilation, linking, and loading. At which step do each of the following events occur?

<table>
<thead>
<tr>
<th>Event Description</th>
<th>Step</th>
</tr>
</thead>
<tbody>
<tr>
<td>Templates are instantiated (eg, vector&lt;int&gt;) for a specific type</td>
<td>Compilation</td>
</tr>
<tr>
<td>Space is reserved for global variables which reside in static data</td>
<td>Linking</td>
</tr>
<tr>
<td>Global variables which reside in static data are initialized to their values</td>
<td>Loading</td>
</tr>
<tr>
<td>The contents of header files (eg, stdio.h) are copied into source (eg, .c)</td>
<td>Preprocessing</td>
</tr>
<tr>
<td>References to declared-but-not-defined symbols (eg, function declarations and extern’ed variables) are resolved</td>
<td>Linking</td>
</tr>
<tr>
<td>Source files (eg, main.cc) are checked for syntax errors</td>
<td>Compilation</td>
</tr>
</tbody>
</table>

Question 4:

UW student numbers (not UWNetIDs) are 7-digit numbers that uniquely identify every currently- and formerly-enrolled student. Unfortunately, the first two digits represent a year, which means the format will need to change in approximately 50 years. UW has decided that the new format for student numbers will be a randomly-generated bit pattern.

If this format needs to last for the next 200 years and there are ~30,000 students per year, what type should you choose to represent these student numbers?

*Hint:* $2^16 = 65,536$; $2^{32} = 4,294,967,296$; $2^{64} = 18,446,744,073,709,551,616$

(A) [3 pts] ☐ Signed integer x Unsigned integer

(B) [3 pts] ☐ 16-bit integer x 32-bit integer ☐ 64-bit integer
Question 5:

This holiday-themed C program has 3 files. Remember that `%` is the modulo or “remainder” operator.

```c
#include "trickortreat.h"
#define NUM_CANDY_TYPES 3
#define TO_CANDY(c) ((c) + 1)

static int kids = 0;

int Dispense() {
    int candy = TO_CANDY(kids % NUM_CANDY_TYPES);
    kids++;
    return candy;
}
```

```c
#include "trickortreat.h"
#include <malloc.h>

void EatCandy(int a[]) {
    for (int i = 0; i < NUM_EATEN; i++) {
        a[i] = EATEN_CANDY;
    }
}

int main(int argc, char *argv[]) {
    int *collectedCandy = (int*)malloc(NUM_PIECES * sizeof(int));
    int kids = 10;
    for (int i = 0; i < NUM_PIECES; i++) {
        collectedCandy[i] = Dispense();
    }
    EatCandy(collectedCandy);
    free(collectedCandy);
    return 0;
}
```

(A) [8 pts] Below, write the contents of `trickortreat.c` after it has been pre-processed.

```c
int Dispense();
static int kids = 0;
int Dispense() {
    int candy =
        ((kids % 3) + 1);
    kids++;
    return candy;
}
```
(B) [20 pts] Draw a memory diagram showing the state of the program at “*** HERE ***”. For your convenience, our two .c files are reprinted below.

```c
#include "trickortreat.h"
#define NUM_PIECES 4
#define NUM_EATEN 3

void EatCandy(int a[]) {
    for (int i = 0; i < NUM_EATEN; i++) {
        a[i] = EATEN_CANDY;
    }
}

int main(int argc, char *argv[]) {
    int *collectedCandy = (int*)malloc(NUM_PIECES * sizeof(int));
    int kids = 10;
    for (int i = 0; i < NUM_PIECES; i++) {
        collectedCandy[i] = Dispense();
    }
    EatCandy(collectedCandy);
    // *** HERE ***
    free(collectedCandy);
    return 0;
}
```

```c
#include "trickortreat.h"
#define NUM_CANDYTYPES 3
#define TO_CANDY(c) ((c) + 1)

static int kids = 0;

int Dispense() {
    int candy = TO_CANDY(kids % NUM_CANDYTYPES);
    kids++;
    return candy;
}
```
Question 6:
Consider the following C++ program:

```cpp
void embiggen(int a[], int size) {
    for (int i = 0; i < size; ++i) {
        a[i] += 1;
    }
}

int main(int argc, const char *argv[]) {
    int arr[] = {0, 10, 20, 30};
    int i = arr[0];
    i += 3;
    int &r = arr[1];
    r += 2;
    int *p = &(arr[2]);
    p += 1;
    embiggen(arr, 4);
    // *** HERE ***
    return 0;
}
```

[19 pts] When this program reaches “*** HERE ***”, what do each of these expressions evaluate to?

<table>
<thead>
<tr>
<th>Expression</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>i</td>
<td>3</td>
</tr>
<tr>
<td>r</td>
<td>13</td>
</tr>
<tr>
<td>*p</td>
<td>31</td>
</tr>
<tr>
<td>arr</td>
<td>{1, 13, 21, 31}</td>
</tr>
<tr>
<td>&amp;i == &amp;(arr[0])</td>
<td>True    False</td>
</tr>
<tr>
<td>&amp;r == &amp;(arr[1])</td>
<td>True    False</td>
</tr>
<tr>
<td>&amp;r == &amp;(arr[3])</td>
<td>True    False</td>
</tr>
<tr>
<td>p == &amp;(arr[2])</td>
<td>True    False</td>
</tr>
<tr>
<td>p == &amp;(arr[3])</td>
<td>True    False</td>
</tr>
</tbody>
</table>
Question 7:

Our templated “Smart Vector” class stores pointers to dynamically-allocated objects and releases their memory when it goes out of scope. Furthermore, it implements “deep copy” semantics by copying the pointers (ie, copying raw memory addresses) whenever a SmartVector is copied.

(A) [10 pts] Implement SmartVector’s copy constructor.

```cpp
SmartVector(const SmartVector &other) {
    currentSize_ = other.currentSize_; 
    for (int i = 0; i < other.currentSize_; i++) {
        contents_[i] = new T( *(other.contents_[i]) );
    }
}
```

(B) [4 pts] SmartVector currently works on any T. Based on your new copy constructor, what restrictions now apply to T’s functionality? If there are changes, describe them below.

- [ ] There Are New Restrictions (described below)
- [x] No New Restrictions

T needs to support copy-construction.
(C) [8 pts] Considering all we know about classes and deep copies, what is `SmartVector` missing and why does it matter?

`SmartVector` doesn’t comply with the “Rule of 3”; it needs to implement an assignment operator to avoid making shallow copies of its contained pointers.

If a `SmartVector` “b” is assigned to a `SmartVector` “a”, then both of them will attempt to delete the same contents when they go out of scope. This will result in a double-delete.

(D) [3 pts] Using 3 lines or fewer, write code that demonstrates the missing functionality discussed in (C). We’ve given you some starter code.

```cpp
#include "SmartVector.h"

int main(int argc, const char * argv[]) {
  SmartVector<int> v1;
  v1.Append(new int(351));
  v1.Append(new int(333));

  __ SmartVector<int> v2; __________

  __ v2 = v1; _________________________

  ________________________________
  return 0;
}
```

**Question 8:**

[1 pt; all non-empty answers receive this point] Select one member of the course staff. Describe or draw an emoji representing that person.

💩 Congratulations on finishing the midterm!