Question 3: It's Fight or FLIGHT [24 pts]

We're rewriting airport software to help the good folks at Sea-Tac keep track of flights. We will use the following typedef-ed structs:

```
typedef struct t {
  size_t hr;   // 0-23
  size_t min;  // 0-59
} Time;
```

```
typedef struct f {
  char *dest;  // destination airport
  Time dep;    // departure time
  Time arr;    // arrival time
} Flight;
```

```
typedef struct a {
  char *name;
  Flight *flights;  // address of array of flights
  size_t num_f;     // number of flights in array
  struct a *next;
} Airport;          // node in linked list of airports
```

Each airport node holds a pointer to an array of `Flights` on the Heap and the length of that array is stored in `num_f`. **Pointers `name` and `dest` should also point to the Heap.** Assume we have the code shown below:

```
// Creates a new Airport (name: copied from argument, flights: NULL,
// num_f: 0, next: NULL) on the Heap and then pushes it to the front
// of our linked list of airports.
Airport *MakeAirport(char *name);

// Takes the provided Flight data and stores it in the flights array
// of the specified airport.  Needs to update flights and num_f.
void AddFlight(Airport *a, char *dest, Time dep, Time arr);
```

```
Airport *head = NULL;

int main(int argc, char **argv) {
  Time t1 = {10,40}, t2 = {12,42};
  head = MakeAirport("SEA");
  AddFlight(head, "SFO", t1, t2);
  head = MakeAirport("SFO");
  return EXIT_SUCCESS;
}
```

SID: _____

(A) Draw a memory diagram for our linked list of airports before `main()` returns: [9 pt]

(B) Complete the implementation of `AddFlight()`. Assume `stdio.h`, `stdlib.h`, and `string.h` are included. Assume arguments are valid, but check for other errors. [15 pt]

```
void AddFlight(Airport *a, char *dest, Time dep, Time arr) {
  // make more space for larger array









  // allocate space for destination name




  // copy data into appropriate fields



  // update number of flights

}
```

Question 3: C Preprocessor [6 pts]

Consider the following code in a file called `spooky.c`:

```
#include <stdlib.h>
#include <stdint.h>
#include <stdio.h>

#define TRIPLE(x) x * 3
#ifdef DDEBUG
#define CONST 4
#else
#define CONST 2
#endif

int main(int argc, char **argv) {
  int32_t x = TRIPLE(2);
  int32_t y = TRIPLE(10 + CONST);
  printf("%d, %d\n", x, y);
  return EXIT_SUCCESS;
}
```

When compiled with `gcc -Wall -std=c11 -DDEBUG -o spooky spooky.c`, what does the executable `spooky` output when run?

Question 6. (20 points) Not quite the traditional what-does-it-print question. Consider the following C++ program, which, as is usual, compiles and executes with no errors.

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

int f(int &n, int *pa, int &k, int *pb) {
  k = pb[1];
  pb[2] = pa[1];
  n = *pb**pa;
  return k+1;
}

int main() {
  int a = 1;
  int &b = a;
  int ray[4] = { 10, 11, 12, 13 };
  int *p = ray;
  int *q = &ray[1];

  *p = f(ray[2], p, b, q);

  cout<< "a = " << a << ", b = " << b << ", *q = " << *q << endl;
  cout<< "ray = ";
  for (int k = 0; k < 4; k++)
    cout << ray[k] << " ";
  cout << endl;

  return 0;
}
```

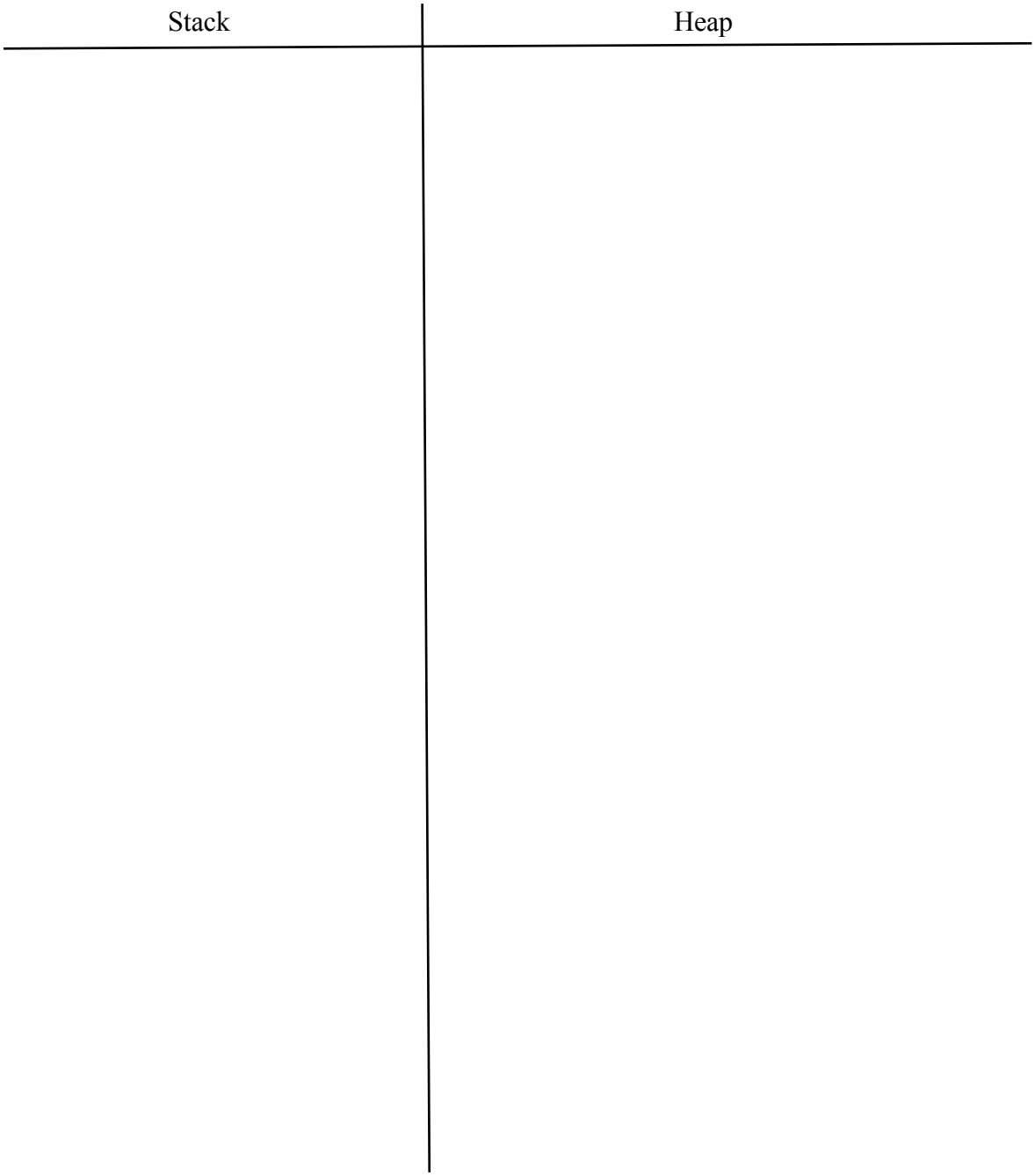
What output does this program produce when it runs? (You are not required to draw a boxes-n-arrows diagram, but you might find doing so to be very helpful, and it might help us if we need to assign partial credit to a not-completely-perfect answer.)

Question 5. (26 points) One of the summer interns is trying to learn C++ and has written the following class that stores an array of doubles and a main program that uses it.

```
class Doubles {
public:
  // construct Doubles given array and # elements
  Doubles(double *vals, uint32_t size)
         : v_(new double[size]), sz_(size) {
    for (uint32_t k = 0; k < size; k++)
      v_[k] = vals[k];
  }

  // destructor, other standard operations
  ~Doubles() { delete[] v_; }
  Doubles(const Doubles &other) = default;
  Doubles &operator=(const Doubles &other) = default;

  // "getter" functions
  double   get(uint32_t k) const { return v_[k]; }
  uint32_t length() const        { return sz_; }

private:
  double* v_;     // heap-allocated array
  uint32_t sz_;   // size of array
};

// print data in a Doubles object
void prdbl(Doubles d) {
  ///// ***>>>> here <<<<*** /////
  cout << "[ ";
  for (uint32_t k = 0; k < d.length(); k++)
    cout << d.get(k) << " ";
  cout << "]" << endl;
}

int main() {
  double a[] = { 1.1, 2.2, 3.3 };
  Doubles d3(a,3);
  Doubles* dp = new Doubles(d3);
  prdbl(d3);
  prdbl(*dp);
  delete dp;
  return 0;
}
```

Please answer the questions about this class on the next page and **remove this page from the exam. This page will not be scanned for grading.**

Question 5. (cont.) (a) (12 points) Draw a *precise* diagram showing the contents of memory the **first time** execution reaches the comment ///// ***>>>> here <<<<*** ///// at the beginning of function `prdbl`. Your diagram should clearly show the contents of the individual stack frames for `main` and `prdbl` and the contents of heap storage, with appropriate arrows from pointers to values that they reference. Then continue with the question on the next page.



(continued on next page)

Question 5. (cont.) (b) (3 points) When the program is executed it crashes. Exactly where does it crash, when, and why? (what is the problem?) (Be brief but precise!)

(c) (3 points) Our summer intern has been googling and thinks that something called the "Rule of 3" is the reason for the crash. The intern proposes replacing the destructor with the following code to match the copy constructor and assignment:

```
~Doubles() = default;
```

Will the program run without crashing if this is done? Why or why not? (briefly)

Question 5. (cont.) (d) (8 points) What really needs to be done to fix this class so it works properly and behaves appropriately for a C++ class? Give the changes needed below by listing which functions (methods) need to be changed in the original code and writing the correct code below.

Question 1: You MAKE Me Whole [12 pts]

For the following questions, you may use the variable `CFLAGS = -Wall -g -std=c11`.

(A) We have a file `oneA.c` that includes `oneA.h`. Write a Makefile target to produce the executable `oneA`. [3 pt]

Recall that targets can execute multiple commands. The `touch` command updates the timestamp on a file to the current time (and creates the file if it did not previously exist).

(B) Draw out a corresponding directed acyclic graph for the Makefile on the left. [4 pt]

```
cse: cse.o engr.o
     gcc $(CFLAGS) -o cse *.o

cse.o: cse.c engr.h uw.h
     touch engr.o
     gcc $(CFLAGS) -c cse.c

engr.o: engr.c engr.h
     gcc $(CFLAGS) -c engr.c

uw.o: uw.c uw.h
     gcc $(CFLAGS) -c uw.c

clean:
     rm -f *.o *~ cse
```

(C) A likely dependency error should be apparent from part B. Describe the fix. [2 pt]

(D) Even with the dependency fix from part C applied, running `make clean` then `make` results in a linking error! *Briefly* describe why this happens. [3 pt]

Question 5: C File I/O [14 pts]

a) [6 pts] Suppose you are using the C standard library to write 1024 total bytes to disk. To do so, you invoke the fwrite() function repeatedly, writing N bytes each time until all 1024 bytes are written.

Give a value of N such that the code would be more efficient with buffering turned on than turned off (assume a buffer size of 512 bytes). Briefly explain why.

Give a value of N such that the code would be more efficient with buffering turned off than turned on. Briefly explain why.

b) [8 pts] Suppose you have a file on disk called `midterm_soln.txt` with a guaranteed size of exactly 50 bytes. The following is a partially-implemented POSIX read loop that reads in those 50 bytes from the file and copies them into a buffer called `buf`. Complete the code so that after the last line, `buf` contains those 50 bytes. Make sure to clean up the open file descriptor.

```
#include <fctnl.h>
#include <unistd.h>
#define BUFFER_SIZE 50

char buf[BUFFER_SIZE];
int bytes_left = BUFFER_SIZE;
int fd = open("midterm_soln.txt", O_RDONLY);

while (1) {
  ssize_t res = read(fd, _____________________________, _____________________);
  if (res == 0) {
    break;
  } else if (res == -1) {
    if (errno != _____________) {
      perror("read error");
      exit(1);
    }
  } else {

    ____________________________________________________________________;
  }
}
_____________________________________; // Clean up the fd
```

Question 5. (16 points) Constructor madness. Consider the following C++ program which does compile and execute successfully. On the next page, write the output produced when it is executed.

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

static int idnum = 1;   // global var: next obj id number

class obj {
public:
  obj() {                    // default constructor
    id_ = idnum; idnum++;
    cout << "obj " << id_ << ": default constructor" << endl;
  }
  obj(int n) {               // int constructor
    id_ = idnum; idnum++;
    cout << "obj " << id_ << ": int constructor" << endl;
  }
  obj(const obj & other) {  // copy constructor
    id_ = idnum; idnum++;
    cout << "obj " << id_ << ": copy constructor from " <<
                                              other.id_ << endl;
  }
  obj& operator=(const obj & other) {  // assignment operator
    cout << "obj " << id_ << ": assignment operator from " <<
                                              other.id_ << endl;
    return *this;
  }
  ~obj() { // destructor
    cout << "obj " << id_ << ": destructor" << endl;
  }
private:
  int id_;    // this obj's id number
};

int main() {
  obj a;         // output is obj 1: default constructor
  obj b(a);
  obj c = 5;
  obj d = c;
  a = c;
  b = 5;
  cout << "done!" << endl;
}
```

Please write your answer on the next page and **remove this page from the exam. This page will not be scanned for grading.**

(continued on next page)

Queston 5 (cont.) On this page, write the output produced when the program from the previous page is executed. It does compile and execute successfully.

Note that when an object is constructed, the constructor stores a unique integer `id_` number, and operations on each object print out that object's `id_` number when they are executed. The first object's `id_` number is 1, and each new object has an `id_` number that is 1 greater than the previous object.

Also note that the constructors and assignment operations ignore their arguments. That, of course, would not happen in real code, but for this question it was done to save space since the values of the arguments are not needed to trace the program's execution.

The first output line is written for you. Write the rest of the program's output after that.

Output:

```
obj 1: default constructor
```