University of Massachusetts - Boston **Kevin Amaral**
Programming in C **Exam #2** **CS 240 – Spring 2018**

Name: _____

Short Questions (20 points)

1. (2 points) When malloc and free are called intermittently with each other, what memory phenomenon is likely to occur?

2. (2 points) Give an example UNIX command for linking two object files `obj1.o` and `obj2.o` into a program `prog`.

3. (2 points) Give an example of a prototype for a function that returns an `int` value and takes two `char` arguments.

4. (2 points) Give an example of why or when you'd want to use a `void*` variable.

5. (4 points) What is Conditional Inclusion and what is one problem that it solves?

6. (4 points) What is the difference between `break` and `continue` in a loop? When should you use `goto`?

7. (4 points) When is a recursive function tail recursive? Why do we want recursive functions to be tail recursive?

Evaluation Questions (20 points)
What are the values of each expression? Account for side-effects. (Problem 1 affects the memory that Problem 2 works with, and so on)

```
int  iarray[] = { 00, 10, 20, 30, 40};
char carray[] = {'0','1','2','3','4'};
struct address {
     char *cptr;
     int *iptr;
};

struct address addr;
struct address *sptr;
int offset;

addr.iptr = iarray;
addr.cptr = &carray[3];
sptr      = &addr;
```

1. `*addr.iptr++`

2. `++*addr.iptr`

3. `*--sptr->cptr`

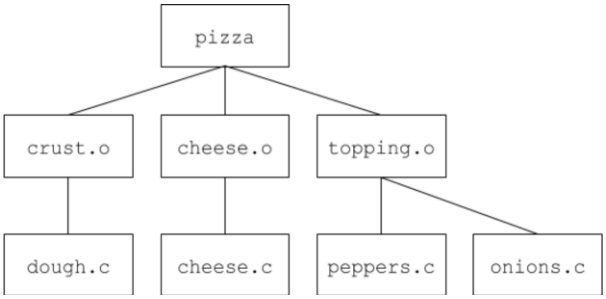
4. `offset = sptr->cptr - carray`

5. `*(iarray + offset)`

Name: _____

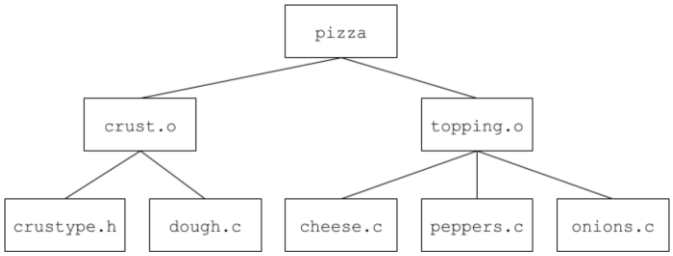
Make File (20 points)

Your company has tasked you and your team with writing a pizza application in C. After further clarification, it seems they wanted you to build an actual pizza, rather than a point-of-sale application. Your team came up with the following break-down of source files for the application.



(10 points) Write a `makefile` for this version of the `pizza` program.

Your boss pulled in some extra hours and reorganized the program structure. They argued that cheese is a topping and it belongs in the topping module. They also added functionality for thin and stuffed crust pizza.



Your boss remade the `makefile` and rebuilt the pizza, however, your boss is a very bad coder. This morning, you worked until noon to rewrite the `dough.c` and `crustype.h` files.

(10 points) Which rules in the new `makefile` need to be rerun to produce the most up-to-date version of `pizza`?

University of Massachusetts - Boston **Kevin Amaral**
Programming in C **Exam #2** **CS 240 – Spring 2018**

Name: _____

Program on Paper (40 points)
You're tasked with writing a program for the security team at a high-profile art gallery. The police will regularly notify the gallery of the names of art thieves. Your job is to keep track of the last 10 patrons to enter the gallery and when they entered (as an integer "unix time"), as well as to keep a list of the last 50 art thief names the police have given you.

Most of the code has been written by your coworkers. Your colleagues even recommended using this struct for your part of the program:

```
struct patron_entry {
    char * patron;
    int time;
}
```

Your part is to write the following functions as well as any necessary global variables:

```
void add_patron_to_list(char * patron, int time);
int add_thief_to_list(char * thief);
```

Specifications for the whole part
1. Must keep track of two separate lists for patrons and thieves.
2. As more patrons arrive, the oldest parton is to be "forgotten" from the list (like the tail program)
3. Thieves should never be forgotten.
4. You may use any string and library functions we have discussed in class.
5. Work on one function at a time and spent your time wisely.

Specifications for `add_patron_to_list`, called when a patron enters the gallery.
1. `patron` variable is an input buffer that will be overwritten with the program's next line of input.
    a. This function should allocate new memory for a copy of the patron's name.
2. When there have been more than 10 patrons, the program should deallocate (free) memory that was allocated from previous patrons.

Specifications for `add_thief_to_list`, called when the police tip you off about a thief.
1. `thief` variable is an input buffer that will be overwritten with the program's next line of input.
    a. This function should allocate new memory for a copy of the thief's name.
2. Function should return `THIEF_SUCCESS` when it successfully adds a thief to the list, and `THIEF_FAILURE` when there are already 50 thieves in the list.
    a. Assume that these are pre-defined symbolic constants with integer values.

Extra Credit:
- When a patron whose name is on the thief list enters the gallery, `call_the_cops()`

University of Massachusetts - Boston **Kevin Amaral**
Programming in C **Exam #2** **CS 240 – Spring 2018**

Name: _____

Answer Key

Short Answers

1. Memory Fragmentation
2. `gcc -o prog obj1.o obj2.o`
3. `int func(char one, char two);`
4. It is a placeholder type for an address to a region in memory where something arbitrary is. Alternatively, it's used to return a generic value where the program can't know ahead of time the return type of a function (i.e. malloc)
5. Conditional inclusion is a way to selectively include or exclude code with precompiler directives #if, #ifdef, etc.
6. break immediately stops the current iteration of the loop and exits the loop whereas continue immediately stops the current iteration of the loop but continues onto the next one.
   Goto should be used when there is no reasonable way to leave a control-flow otherwise in a nice way. Alternatively, if you simply need to go to a specific part of the code and there's no better way to do so.
7. A tail recursive function is a recursive function in which the very last thing done in the function is the recursive call.
   The reason we want recursive functions to be tail recursive is that it allows the compiler to reduce the amount of function stack frames on the stack. This solves one of the downsides of recursive functions.

Evaluation Questions

1. 0
2. 11
3. '2'
4. 2
5. 20

Makefile

Part 1

```
pizza:  crust.o cheese.o topping.o
        gcc -o pizza crust.o cheese.o topping.o

crust.o:        dough.c
                gcc -c -o crust.o dough.c

cheese.o:       cheese.c
                gcc -c -o cheese.o cheese.c

topping.o:      peppers.c onions.c
                gcc -c -o topping.o peppers.c onions.c
```

Name: _____

Part 2

The rules that need to be rerun are as follows:
The crust.o rule is the only rule that directly depends on any of the two changed files.
Then, pizza is the only rule that depends on any of the other rules that need to be rerun.
There's nothing else with dependencies that need to be rerun.

crust.o and pizza

Program on Paper

```
struct patron_entry people[10];
int head  = 0;
int tail  = 0;
int count = 0;

void add_patron_to_list(char * patron, int time) {
     struct patron_entry person;
     int i;

     person.patron = (char *) malloc(strlen(patron));
     for (i = 0; patron[i]; i++) // (i < strlen(patron)) also works as a condition
          person.patron[i] = patron[i];
     person.patron[i] = '\0';

     person.time = time;

     if (head == tail && count == 10)
          free(people[tail].patron);

     people[tail] = person;
     if (head == tail && count == 10) {
          head = (head + 1) % 10;
          tail = (tail + 1) % 10;
     } else {
          tail = (tail + 1) % 10;
          count++;
     }
}
```

Name: _____

```
char *thieves[50];
int thiefcount = 0;

int add_thief_to_list(char * thief) {
     if (thiefcount == 50)
          return THIEF_FAILURE;

     thieves[thiefcount++] == (char *) malloc(strlen(thief));

     for (i = 0; thief[i]; i++) // (i < strlen(thief)) also works as a condition
          thieves[thiefcount][i] = thief[i];
     thieves[thiefcount][i] = '\0';

     return THIEF_SUCCESS;
}
```