Name: ____

University of Massachusetts - Boston **Dr. Ronald Cheung**
Programming in C **CS 240 - Spring 2011**

In-Class, Open Book Exam II
April 28, 2011

The work on this examination is to be your own and you are expected to adhere to the UMass-Boston honor system. All questions can be answered by one or two short sentences. Do not try to make up for a lack of understanding by providing a rambling answer.

Note: I give partial credit! Show all work!

1. (20 points) UNIX and C

a. (2 points) If array abc is defined as char abc[10], what is wrong with abc=abc +1?

b. (2 points) What does the UNIX command tail do?

c. (2 points) Successive operations of malloc and free can cause ______________ in memory.

d. (2 points) What is the major use of the generic(or void) pointer?

e. (2 points) What is the difference between a union and a struct?

f. (2 points) Give an example on how to terminate a recursion.

g. (4 points) char a[10], b[] = "ABCD";

value returned from strlen(b): ____________

how many chars do strcpy(a,b) copy? ______

h. (2 points) Why is it not a good idea to define 1 big common header file to be used by all functions?

i. (2 points) What is a macro?

2. (20 points) Evaluate expressions

Fill in the values. Assume the pointer gets updated after each evaluation.

```
char  text[] = "This_is_a_very_easy_test";
char phone[] = "6172876483";
struct tag{
  char *p2;   char *p3;} name ={phone,text};

char *p4=name.p3;
struct tag * p5=&name;
```

value of *p4++ ________

value of ++*p4 ________

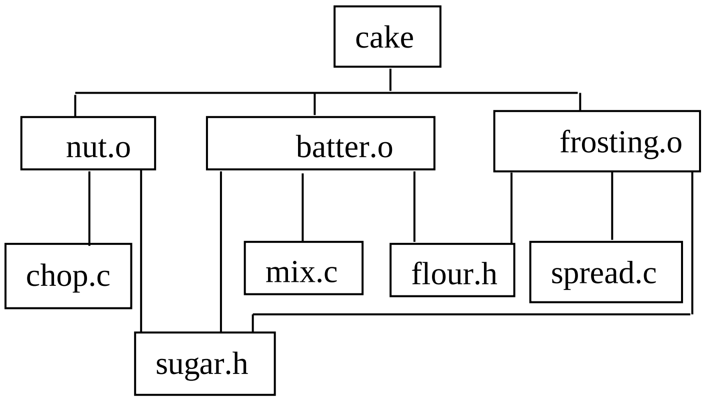
value of p4[2] ________

value of p5->p2 ________

value of * ++p5->p2 ________

CONTINUE ON REVERSE SIDE

3. (20 points) makefile



The dependency list of making cake is shown in the above diagram.

i) (10 points) Write a makefile to make cake. Include a clean option to delete all object files.

ii) (10 points) If you use the UNIX command "touch" to modify file flour.h, and re-make cake, write down what make will do.

4. (40 points) logging function

You are asked to write a C function log_error() to log error messages from a main program. The error message will have a time stamp and a text string. The time stamp will be expressed in a 24-hour integer format (e.g. 8:20 am will be stored as 0820 and 8:20pm will be 2020). The function will compose the message (time stamp + text) and store in a buffer which only has room for 50 total messages. It is required to log the most recent 50 messages. The function will print out all 50 messages stored at the end.

The main program is as follows:

```
int main(){
  int time_stamp;
  char err1[]= "error: buffer overflows!";
  char err2[]= "error: divide by 0!";
  ….
  time_stamp=0820;
  log_error(time_stamp, err1);
  …
  time_stamp=2020;
  log_error(time_stamp, err2);
  …
}
```

Prototype for the log_error function is as follows:

```
int log_error(int ts, char * msg_ptr);
```

where ts is the time stamp,
msg_ptr points to the text of the error message,
and the function returns 0 if failure occurs.

Show your pseudo-code and C code.

Answers:

1.

a. abc =&abc[0] is an address determined during compilation/linking. Cannot be modified during execution.

b. tail command prints out the last n lines of a file

c. fragmentation in memory.

d. Library functions can return a generic pointer(void *) to calling programs and the calling programs can use it by casting it to the corresponding type.

e. A struct/union is a collection of variables grouped under 1 name. A union's members share the memory of the largest member whereas a struct has different memory assigned to the members.

f. Define a recursion as follows:

```
void foo(int j){
    j--;
    if(j >0) foo(j);
}
```

Recursion terminates when j $\leq$0.

g. 4, 5

h. Changing variables used by functions in 1 file cause a recompilation of functions in all files.

i. A macro is a direct character substitution and it does not understand C expressions.

2. pointer gets updated after evaluating the expression:

'T'	(de-reference p4 and inc p4 afterwards)
'i' or 'h'+1	(de-reference p4 and incr. the value)
's'	(p4[0]='h', p4[0+2]='s')
&phone[0]	(pointer value of p2)
'1'	(incr. pointer and de-reference it)

3.

i)

```
cake: nut.o batter.o frosting.o
        gcc nut.o batter.o frosting.o –o cake
nut.o: chop.c sugar.h
            gcc –c –o nut.o chop.c
batter.o: mix.c sugar.h flour.h
             gcc –c –o batter.o mix.c
frosting.o: flour.h spread.c sugar.h
              gcc –c –o frosting.o spread.c
clean:
              rm *.o
```

ii)

```
gcc –c –o batter.o mix.c
gcc –c –o frosting.o spread.c
gcc nut.o batter.o frosting.o –o cake
```

4.

<u>Pseudo code for function log-error begins here:</u>

```
Form the err  message by combining time_stamp and text
Allocate buffer long enough to store err message
Check pointer and print error if pointer ==NULL
Store err message in buffer.
Call function update_buffer to store pointers in a circular buffer
Call function print_buffer to print buffer starting from the beginning of buffer
Return
```

<u>Pseudo code for function update_buffer begins here:</u>

```
Check to see if the pointer array slot is empty
If it is not,
    Free the msg pointer slot
Save the new msg pointer in the present slot.
Inc the pointer and check for wrap around.
If it has wrapped around, set pointer =0
```

<u>Pseudo code for function print_buffer begins here:</u>

```
Starting from 0 to the end of the buffer, print the lines
```

```
/* Exam 2 - Error Logging program starts
here
main -            for testing error_log
                  function
error_log -       to log  MAX_BUFFER error
                  messages
update_buffer -  insert new error messages
                  in buffer
print_buffer -  print all MAX_BUFFER lines
*/

#include <stdio.h>
#define MAXLINE 100
#define MAX_BUFFER 50

int new_index;
char * ptr_array[MAX_BUFFER];
void update_buffer(char * new);
int log_error(int , char *);
void print_buffer(void);

main()
{
  int time_stamp=0;
  char line[MAXLINE];

/* input line has format: time_stamp
Message */
  while(scanf("%d %s",&time_stamp,
        line)!=EOF)
  {
    if (log_error(time_stamp, line) ==0)
      printf("memory allocation error\n");
   }
}


/* function to log error messages
    return 0 if function fails */
int log_error(int ts, char *msg_ptr)
{
    char temp[MAXLINE];
    char *ptr;

/* form the message and store in a buffer */
    sprintf(temp, "%d %s",ts,msg_ptr);

/* allocate buffer: add 1 byte for '\0'*/
    ptr = (char *) malloc(strlen(temp)+1);
    if (ptr == NULL){
        printf("no more memory\n");
        return 0;
    }
    else{
        strcpy(ptr,temp);
        update_buffer(ptr);
        print_buffer();
        return 1;
            }
       }

/* code to print error messages starting
   from the beginning */
void print_buffer(void)
{
   int i, temp_new_index;
   temp_new_index = new_index;
   for (i = 0; i < MAX_BUFFER; i++){
     if(ptr_array[temp_new_index] != NULL)
     printf("%s\n",
        ptr_array[temp_new_index]);
     temp_new_index =(temp_new_index +1)%
        MAX_BUFFER;
   }
}

/* function to update the buffer with new
   error messages */
void update_buffer(char * new)
{
   if( ptr_array[new_index] != NULL)
   {
       free(ptr_array[new_index]);
       ptr_array[new_index] = NULL;
   }
   ptr_array[new_index] = new;
   new_index ++;
   if (new_index >= MAX_BUFFER)new_index= 0;
}
```