EN 601.220 Intermediate Programming, Fall 2019
Midterm Exam

Name: Solution JHED ID: ____________

This test is closed-book, closed-notes, closed-computer. No Internet access or collaboration of any kind is permitted. Print legibly and strongly enough for scanned copies to be easily read.

"I agree to complete this exam without unauthorized assistance from any person, materials or device."

Sign: ____________ Date: ____________

NOTES:

The total number of points possible on the exam is 100, and you will have 75 minutes to complete it.

For all questions on this exam, you should assume the following:
- sizeof(void*) == 8
- sizeof(char) == 1
- sizeof(int) == 4
- sizeof(float) == 4
- sizeof(double) == 8

Also, in the code you are asked to write, you can assume that all dynamic memory allocations succeed.

Testing tips: Read through the entire exam first to get an idea of what is there. Look at the hard problems initially to put them in your head, but do the quicker problems first (quickly), leaving time for the longer ones.

CODE WRITING. [40 pts total: 2 questions worth 20 pts each]
Complete the indicated parts (i.e., steps) in the main() function to perform specified tasks. You need NOT show `#include` statements used by your function. You may assume that all calls to malloc/realloc/calloc succeed.

1) Given the following struct and function definitions:

```
typedef struct {
    int * ages;  // array to store age numbers as int values
    float * weights;  // array to store weights in lb as float values
} AgeWeight;

void print(const AgeWeight data, int size) {
    for (int i = 0; i < size; i++)
        printf("%d %.2f\n", data.ages[i], data.weights[i]);
}
```

Complete the main function in the next page so that it reads data from the user until the end of input is indicated, storing all the data (indirectly) into an AgeWeight struct. The function must read from standard input, and you can assume that the input alternates age (an integer) and weight (a float) values without any errors. You may assume that age numbers are always valid integer values and weights are also valid float values. You should initially dynamically allocate enough memory space for 2 pairs of data. Each time the arrays fill up as the function reads from input and you need more space, increase the capacity by adding memory space to allow 2 more data pairs. The size variable should hold the actual number of pairs of values read (ie, how many pairs of ages and weights were successfully stored). We've also provided a `print` function that should successfully output all the data if called with `print(data);`. (You are not expected to call `print`; it's just for reference, to help you understand `AgeWeight`'s structure.). At the end of the main function, do not forget though to de-allocate any dynamically allocated memory in your program so that you will not have any memory leakages.

[Hint: draw a picture of what should be happening in memory before you begin.]

```
int main(void) {
   AgeWeight data; // a struct to store pairs of age and weight data
   int size = 0; // used to keep track of the number of data pairs

   printf("enter ages and weights, ^d to end\n");

    data.ages = malloc(cap * sizeof(int));
    data.weights = malloc(cap * sizeof(float));
    int a = -1;
    float w = -1.0;
    while (scanf (" %d%f", &a, &w) == 2) {
      if (size == cap) {
        cap += 2;
        data.ages = realloc(data.ages, cap * sizeof(int));
        data.weights = realloc(data.weights, cap * sizeof(float));
      }
      data.ages[size] = a;
      data.weights[size] = w;
      size++;
    }
    print(data, size);
    free(data.ages);
    free(data.weights);
```

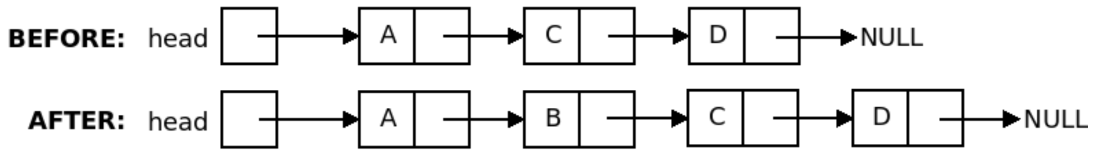
2) Recall the definition of the `Node` data type and the declaration of function `create_node` below:

```
typedef struct node_ {
      char data;
      struct node_ * next;
} Node;
```

// returns pointer to newly-created node with data equal to val and next equal to NULL

```
Node * create_node(char val);
```

Making use of the `Node` definition and `create_node` function shown above, fill in the definition of the function `insert_before` below so that it behaves as specified. In your code, assume that `create_node` always returns a valid pointer (i.e. malloc does not fail). The following diagram shows the expected behavior of a call `insert_before(&head, 'C', 'B')`:



Hints: Draw pictures of example linked lists and make sure operations are sequenced correctly. Think about what should happen when inserting a new first node.

```
// Insert a new node containing the data value dval before the first occurrence
// of a node containing the data value sval.  The parameter list_ptr points to the pointer
// variable which points to the first node in the list.  If there are no occurrences of sval in
// the list, the function should return without modifying the list.
void insert_before(Node **list_ptr, char sval, char dval) {
  // if empty list, do nothing
  if (*list_ptr == NULL) {
    return;
  }

  // special case for inserting before first node
  if ((*list_ptr)->data == sval) {
    Node *n = create_node(dval);
    n->next = *list_ptr;
    *list_ptr = n;
    return;
  }

  // general case:
  // prev lags cur, at each step see if cur contains sval,
  // if so insert a node between prev and cur
  Node *prev = *list_ptr, *cur = (*list_ptr)->next;
  while (cur != NULL) {
    if (cur->data == sval) {
      Node *n = create_node(dval);
      n->next = cur;
      prev->next = n;
      return;
    }
    prev = cur;
    cur = cur->next;
  }
```

CODE TRACING. [6 pts total: 2 outputs worth 3 pts each]

3) Consider the `fun1()` definition below.

```
void fun1(char * s1, char * s2) {
  assert(strlen(s1) % 2 == 0);  //crash if length of s2 is not even
  char s3[100];
  int c = 0;
  while (*s1 && *s2) {
    s3[c] = *s1;
    c++;
    s3[c] = *s2;
    c++;
    s1 += 2;
    s2++;
  }
  while (*s1) {
    s3[c] = *s1;
    s1 += 2;
    c++;
  }
  while (*s2) {
    s3[c] = *s2;
    s2++;
    c++;
  }
  s3[c] = '\0';
  printf ("%s\n", s3);
}
```

Show the output of the following calls to the function. If a particular call crashes or could cause an invalid memory access, simply write "crashes".

```
fun1("be", "colorful");
fun1("look", "within");
```

ANSWER:

bcolorful

lwoithin

MULTIPLE CHOICE. [44 pts total: 11 questions, 4 pts each question]

4) What is the value of `((x<<3) | (y>>2)) & z` given that x, y, and z are all integer variables with values: x = 3, y = 14, and z = 15?

A. 15
B. 14
C. -6
D. 10
E. 11

5) What is the output of the following program?

```
#include <stdio.h>
void fun(int x, int *y, int **z) {
      x -= 1;
      (*y) += 6;
      **z = 5;
}
int main() {
      int a = 7;
      int b = 9;
      int * c = &b;
      fun(a, &b, &c);
      printf("%d %d %d", a, b, *c);
}
```

A. 7 9 9
B. 7 5 5
C. 6 9 5
D. 6 5 5
E. The code compiles with no warnings, but the output does not match any of the above
F. The code does not compile and/or has warning messages

6) Consider the following typedef which might be used as part of a linked list construction:

```
typedef struct _node {
      char data;
      struct _node * next;
} Node;
```

Suppose that `Node * head` is a pointer to the first node in a list, declared in a main function.

Which choice is a correct declaration and function call for a linked list's `add_front` function, which is meant to add a new node to the front of the linked list with the value `'A'`?

A) `void add_front(Node * head, char val);`
called by `add_front(*head, 'A');`

B) `void add_front(Node * head, char val);`
called by `add_front(&head, 'A');`
C) `void add_front(Node * head, char val);`
called by `add_front(head, 'A');`
D) `void add_front(Node ** list_ptr, char val);`
called by `add_front(&head, 'A');`
E) `void add_front(Node ** list_ptr, char val);`
called by `add_front(head, 'A');`

7) Given the declaration `int ** x;` which of the following reads in an integer and stores it in the variable pointed to by `x`?

A. `scanf("%d", x);`
B. `scanf("%d", &x);`
C. `scanf("%d", *x);`
D. `scanf("%s", x);`

8) What is the value of `z` after this statement?

```
int z = 7 / 3 - 15 * 3 + (7 % 2);
```

A. `1`
B. `-42`
C. `39.1`
D. `22`

9) Which variables are automatically initialized to 0 when they are declared?
A. all positions in local `int` array variables
B. static variables
C. global variables
D. B and C
E. A, B and C

10) Given the following function definition:
```
void move(int *p, int *q) {
    int r = *p;
    *q = r;
    *q = *p;
}
```
What is output by the following code?
```
int x = 4, y = 9;
move(&x, &y);
printf("%d  %d", x, y);
```

A. 4 9
B. 9 4
C. 9 9
D. 4 4
E. none of the above

11) Given the declaration

```
            int arr[3][3] = {{1, 2, 3}, {4, 5, 6}, {7, 8, 9}};
```

, what is the output of the following code?

```
int l = 0;
for (int i = 0; i < 3; ++i) {
   for (int j = 2; j > 0; --j) {
      if (i + j == 2) {
         l += arr[j][i];
      }
   }
}
printf("%d\n", l);
```

A. 14
B. 15
C. 18
D. 12
E. 9
F. 3

12) What is the output of the following code?

```
#include <stdio.h>
void fun(int * arr) {
  printf ("%d ", (int)sizeof(arr));
  arr[0] = 8;
}
int main() {
  int ray[] = {1, 2, 4, 9};
  fun(ray);
  int sum = 0;
  for(int i = 0; i < (int)sizeof(ray[0]); i++) {
    sum += ray[i];
  }
  printf("%d", sum);
  return 0;
}
```

A. 4 23
B. 4 16
C. 16 16
D. 16 23
E. 8 16
F. 8 23

13) Which of the following statements is <u>true</u>?

1) A double value can be used as an array index; it automatically gets casted to an int
2) Memory chunk can be dynamically de-allocated from stack space using `free`
3) `int * transpose(int list[6][12], int nr);` is an invalid function declaration as we must not provide the number of rows
4) `ip1 = *ip2;` is a valid assignment given that `ip1` is a pointer and `ip2` is a double pointer (i.e., pointer to pointer)
5) None of the above

14) Which one is a correct declaration of main function in a program that correctly accepts command line arguments?

1) `int main(char argc, char *argv)`
2) `int main(int argc, char argv[][])`
3) `int main(char argc, int *argv[])`
4) `int main(char *argv[], int argc)`
5) `int main(char **argv, int argc)`
6) `int main(int argc, char *argv[])`

TRUE/FALSE. [10 pts total: 10 questions, 1 pt each question]
Clearly indicate your choice by <u>completely coloring in</u> the circle next to True or False.

○ True
○ False
	1) The following code prints `length=7`:
	`char str[] = {'f','o','o','\0','b','a','r','\0'};`
	`printf("length=%d\n", (int) strlen(str));`

○ True
○ False
	2) The following code prints `output is 1`:
	`int a[] = { 1, 2, 3, 4, 5 };`
	`int *p = &a[3];`
	`printf("output is %d\n", *(p - 3));`

○ True
○ False
	3) If `ival` is an `int` variable and `fval` is a `float` variable, then the following code is guaranteed to preserve the value of `ival`:
	`fval = ival;`
	`ival = fval;`

○ True
○ False
	4) If `cval` is a char variable and `ival` is an int variable, then the following code is guaranteed to preserve the value of `cval`:
	`ival = cval;`
	`cval = ival;`

○ True
○ False
	5) The following code is guaranteed to print a positive number:
	`unsigned x;`
	`scanf("%u", &x);`
	`x = x + 1;`
	`printf("%u\n", x);`

○ True ○ False	6) If `x` is an int variable storing a positive value, `x >> 1` and `x / 2` compute the same result.
○ True ○ False	7) Memory allocated with `malloc` is automatically released when the function that calls `malloc` returns.

○ True
○ False
	8) The following code is guaranteed to print a positive value:
	`unsigned x = 0;`
	`x--;`
	`printf("%u\n", x);`

○ True ○ False	9) `printf(`*`args`*`)` and `fprintf(stdout, `*`args`*`)` (where *`args`* is a format string and optional arguments) do exactly the same thing.
○ True ○ False	10) If you have a pointer `p` to a node in a linked list, without access to a pointer to the head node, it's possible to remove the node `p` points to.