Cmput 201 Final practice

Practical Programming Methodology (University of Alberta)

Last Name:_____ **ID#:**_____

Question 1 – Bitwise operations (2 marks total)

For each of the following, what is the **hexadecimal** value of `y` after the assignment has been executed? Remember that an unsigned short int is stored in **16 bits**. You must write down your answer in proper hexadecimal format. You must show how you got the answer; otherwise, you will not get the grade.

(a) [**1 mark**]

```
unsigned short int a = 22, y;
y = a | 13;
```

(b) [**1 mark**]

```
unsigned short int a = 0x04cf, y;
y = a << (a & 3);
```

Last Name:_____ **ID#:**_____

Question 2 – Bitwise operations (2.5 marks total)

Implement the following function (using this exact function prototype):

```
unsigned int rotate_right(unsigned int i, int n);
```

`rotate_right` returns the result of shifting the bits in `i` to the right by `n` places, with the bits that were "shifted off" moved to the left end of `i`. For example, the call `rotate_right(0x12345678, 4)` must return `0x81234567` and the call `rotate_right(0xc2345678, 2)` must return `0x308d159e`. You **MUST** use bitwise operators to implement this function. You will not get any marks if you implement it in another way. **Your code must run correctly on any machine, regardless of how many bits it uses to store an `unsigned int`.**

```
unsigned int rotate_right(unsigned int i, int n){




}
```

Last Name:_____ **ID#:**_____

Question 3 (10 marks total)

Write a program that prompts the user for a number `n`. It then proceeds to read an `n*n` array of integers, inputted row by row. After reading the array, it prints the row sums (i.e., each row has a sum which is the sum of all the elements in this row), the column sums (i.e., each column has a sum which is the sum of all elements in this column), and a diagonal sum (which is the sum of all elements on the diagonal ↘). Underlined lines indicate user input. You can assume that your program will be compiled with the C99 standard.

Here is a sample run of the program

```
Enter size:  5
Enter row 1:  8 3 9 0 10
Enter row 2:  3 5 17 1 1
Enter row 3:  2 8 6 23 1
Enter row 4:  15 7 3 2 9
Enter row 5:  6 14 2 6 0

Row totals:  30 27 40 36 28
Column totals:  34 37 37 32 21
Diagonal total:  21
```

Question 4 – Linked Lists (11 marks total)

Consider a linked list that keeps the list of students enrolled in a course. The structure of each Student in the list is as given, and the head of the linked list is defined in a struct called `StudentList`. If the list is empty, then head is NULL.

Write three functions for this question: `createStudentList`, `insertStudent`, and `reverse`.

The details of each function are provided in the sub-question. You must dynamically allocate memory for each node. There is no maximum size for the length of each student name so student names must also be dynamically allocated. **You must handle all corner cases as applicable to each function: e.g., handling empty lists, lists with one node etc.** You can assume that any list passed to `insertStudent` or `reverse` has been created using `createStudentList`.

A main function has also been provided to illustrate how a client program would use these functions.

```
struct Student{
    char *name;
    float grade;
    struct Student *next;
};
```

```
struct StudentList{
    struct Student *head;
};
```

```
int main(){
    struct StudentList *studentList = createStudentList();
    insertStudent(studentList, "Alice", 3.7); //list = Alice
    insertStudent(studentList, "Bob", 3.2); //list = Alice, Bob
    insertStudent(studentList, "Mark", 3.6); //list = Alice, Bob, Mark
    reverse(studentList);//list now = Mark, Bob, Alice
    return 0;
}
```

(a) [**2.5 marks**]

```
//returns a new dynamically allocated empty student list
struct StudentList* createStudentList(){









}
```

(b) [**5 marks**]

```
//This inserts a new student with the given name and grade to the **END of the list**
void insertStudent(struct StudentList *studentList, char *name, float grade){









}
```

(c) **[3.5 marks]**

```
//Reverses the linked list (i.e., make the first node of the list as the last
//node, the 2nd node as the one before last etc. and updates head accordingly
//You MUST implement the reverse function "in-place" i.e., do not create a separate list
//or allocate any new memory
void reverse(struct StudentList *studentList){









}
```

(Empty page)

Last Name:_____ **ID#:**_____

Question 5 – Function Pointers and declarations/program organization (7 marks total)

Consider the following declarations inside `prog.c` for the next four sub-questions.

```
struct employee {
    char name[100];
    char phone[12];
};

struct employee list[200];
```

(a) Write a comparison function called `cmp` to be passed to `qsort` to sort `list` on the `name` member in **ASCENDING** order based on the ASCII values. `cmp` will be defined in `prog.c`.

The following is an excerpt of the man page of qsort. The prototype of qsort also shows the prototype of the comparison function it accepts:

```
void qsort(void *base, size_t nmemb, size_t size,
                int (*compar) (const void *, const void *));

The qsort() function sorts an array with nmemb elements of size size.
The base argument points to the start of the array.

The contents of the array are sorted in ascending order according to a
comparison function pointed to by compar, which is called with two arguments
that point to the objects being compared.

The comparison function must return an integer less than, equal to, or greater than
zero if the first argument is considered to be respectively less than, equal to,
or greater than the second. If two members compare as equal, their order in
the sorted array is undefined.
```

(b) Using the `cmp` function you defined in the previous step, sort all employees in `list` using qsort. Assume that `list` is already correctly initialized with entries.

(c) You want to allow `cmp` to be used by other files that would also like to sort by `struct employee name` members. What is the new declaration of `cmp` in `prog.c` that would allow this?

Last Name:_____ **ID#:**_____

Question 6 – Declarations (6 marks total)

In the provided table, you are asked about certain variables in the given program. For each variable, indicate its type, its storage duration, and circle all program points it is visible/accessible from based on its scope. You must circle all the applicable program points to get the marks for the last column.

```
#include <stdio.h>
int count = 0;

int increment(int x, int j){
    static int visits = 0; //program point p1
    visits++;
    return x+j;
}

int main(){
    int y = 4, *p = &y;
    increment(*p, 2); //program point p2
    printf("y=%d\n", y);

    for(int i = 0; i < 5; i++)
        printf("y=%d\n", increment(y, i)); //program point p3

    //program point p4
    return 0;
}
```

As a reminder, here is a non-exhaustive list of the different kinds of specifiers: `auto, static, extern, register, const, volatile, restrict, void, char, short, int, long, float, signed, unsigned`

Variable	Type (Qualifiers & Specifiers)	Storage Duration	Program points it can be accessed from
`count`			p1 p2 p3 p4
`x`			p1 p2 p3 p4
`visits`			p1 p2 p3 p4
`y`			p1 p2 p3 p4
`p`			p1 p2 p3 p4
`i`			p1 p2 p3 p4

Question 7 – Dynamic Memory Allocation (5 marks total)

The following program contains memory leaks. Fix the program so it contains no memory leaks. **Make sure to indicate where your changes will be added in the program**. You should not change any of the given statements during your fixes.

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

int main(){

    char **strings = malloc(5 * sizeof(char *));

    for(int i = 0; i < 5; i++){
        strings[i] = malloc (30);
        strcpy(strings[i], "hello");
    }

    printf("Array contains:\n");
    for(int i = 0; i < 5; i++){
        printf("%s\n", strings[i]);
    }


    return 0;
}
```

Question 8 – Pointers, arrays, and dynamic memory allocation (14 marks total)

You are given the main function below and are asked to implement **4 functions**. You must decide on the type of the parameters each function will take based on the following description and the `main` function code that shows how these functions will be used. **You MUST NOT change the code in main: your implemented functions MUST work correctly with the function calls in main**

1. [**4 marks**] `initArray` takes (1) **a POINTER to an array of integer pointers** and (2) the number of elements it should allocate. Each element should be initialized to `0`. After calling `initArray` in the main function below, `numbers` will be the memory address of an array of integer pointers.

2. [**4 marks**] `addToArray` takes (1) an array of integer pointers, (2) the number of elements in the array, and (3) an integer pointer. It finds the first available element in the array and makes it point to a **dynamically** allocated space that contains the value of the passed integer. If there are no available elements, it prints an error message to the correct output stream and returns.

3. [**3 marks**] `print` takes an array of integer pointers and the number of elements `n` it should print. It then prints a comma separated list of the first `n` integers stored in the array. The last element is followed by a newline. If one of these elements is `NULL`, it skips over it.

4. [**3 marks**] `swap` takes (1) an array of integer pointers, (2) an index `i1`, and (3) an index `i2`. It then swaps the integers at indices `i1` and `i2`. You can assume that both indices are within the bounds of the array. You **must** implement swap **WITHOUT** allocating any new memory using malloc/calloc/realloc or creating a new array.

```
int main(){
    int **numbers;
    initArray(&numbers, 5); //dynamically allocates memory for nameArray

    //first call adds "Bob" in the first available element in the array
    //(after dynamically allocating memory for Bob)
    int a = 1, b = 2, c = 3, d = 4, e = 5;
    int *p = &a, *q = &b, *r = &c, *s = &d, *t = &e;
    addToArray(numbers, 5, p);
    addToArray(numbers, 5, q);
    addToArray(numbers, 5, r);
    addToArray(numbers, 5, s);
    addToArray(numbers, 5, t);

    print(numbers, 5);   //prints 1, 2, 3, 4, 5
    swap(numbers, 0, 1); //swaps elements 0 and 1
    print(numbers, 5);   //prints 2, 1, 3, 4, 5
    swap(numbers, 1, 3); //swaps elements 1 and 3
    print(numbers, 5);   //prints 2, 4, 3, 1, 5
    swap(numbers, 4, 0); //swaps elements 4 and 0
    print(numbers, 5);   //prints 5, 4, 3, 1, 2
    destroy(&numbers, 5) //assume function exists to free memory
}
```

(Empty page)

(Empty page)

Question 9 (6 marks total)

Given the following C file called inventory.c.

```
#include <stdio.h>
#include <string.h>
typedef int Quantity;
typedef struct {
    char name[20];
    Quantity quantity;
    float price;
} Item;

int main(){
    Item items[3];
    strcpy(items[0].name, "Tomato");
    items[0].quantity = 10;
    items[0].price = 2.0;

    strcpy(items[1].name, "Carrot");
    items[1].quantity = 40;
    items[1].price = 1.7;

    strcpy(items[2].name, "Cucumber");
    items[2].quantity = 0;
    items[2].price = 2.5;

    #ifdef RESTOCK
        for (int i = 0; i < 3; i++){
            items[i].quantity += 100;
        }
    #endif

    #ifdef FULL
        for (int i = 0; i < 3; i++){
            printf("%d %ss at %.2f per piece\n", items[i].quantity,
                items[i].name, items[i].price);
        }
    #else
        for (int i = 0; i < 3; i++){
            printf("%ss at %.2f per piece\n", items[i].name,
                items[i].price);
        }
    #endif
    return 0;  }
```

(a) **[1.5 marks]** What is the output of the program when it is run as `./inventory` after compiling it using the following command: `gcc -Wall -std=c99 -o inventory inventory.c`

(b) **[1.5 marks]** What is the output of the program when it is run as `./inventory` after compiling it using the following command: `gcc -Wall -std=c99 -DFULL -o inventory inventory.c`

(c) **[1.5 marks]** What is the output of the program when it is run as `./inventory` after compiling it using the following command: `gcc -Wall -std=c99 -DRESTOCK -o inventory inventory.c`

(d) **[1.5 marks]** What is the output of the program when it is run as `./inventory` after compiling it using the following command: `gcc -Wall -std=c99 -DRESTOCK -DFULL -o inventory inventory.c`

Question 10

Consider the following declarations and definitions for this question:

```
int grades[100][10];
char *days[] = {"Monday", "Tuesday", "Wednesday", "Thursday",
                "Friday", "Saturday", "Sunday"};
```

(a) [**2 marks**] Write code to initialize all elements of `grades` to zero.

(b) [**2 marks**] Set the element at row 15 and column 5 of `grades` (i.e., grades[15][5]) to 40 using pointer arithmetic and **without using '[' or ']'**. Make sure to use appropriate parentheses, if needed.

(c) [**1 mark**] What is the exact character or string that `days[2][2]` refers to?

(d) [**1 mark**] What is the exact character or string that `days[4]` refers to?

(e) [**1 mark**] What is the exact character or string that `*(days[6] + 2)` refers to?

(f) [**1 mark**] What is the exact character or string that `*(days + 3)` refers to?

Question 11 – Program Design

Implement your own `strlen` function called `my_strlen`. You can not use character comparisons, but are allowed to use the following functions: `strcmp`, `strcpy`, `strcat`. (Note that you don't *need* to use these functions.) According to the man page definition for `strlen`:

The strlen() function calculates the length of the string pointed to by s, excluding the terminating null byte ('\0').

You can not access individual characters of the string (using `[]`), but are allowed to use pointers to access sub-strings. As a reminder, here is the signature of the original `strlen` function, and the signatures for the other three string functions you **can** use (along with their descriptions):

```
size_t strlen(const char *s);
```

```
int strcmp(const char *s1, const char *s2);
```

`strcmp()` returns an integer indicating the result of the comparison, as follows:

- *0, if the s1 and s2 are equal;*
- *a negative value if s1 is less than s2;*
- *a positive value if s1 is greater than s2.*

```
char *strcpy(char *restrict dst, const char *restrict src);
```

This function copies the string pointed to by src, into a string at the buffer pointed to by dst. The programmer is responsible for allocating a destination buffer large enough, that is, strlen(src) + 1.

```
char *strcat(char *restrict s1, const char *restrict s2);
```

The strcat() function shall append a copy of the string pointed to by s2 (including the terminating NULL character) to the end of the string pointed to by s1. The initial byte of s2 overwrites the NULL character at the end of s1.

(Empty page)