Fundamentals of Computing and Programming, 1 Semester, 2023-24

Back paper Exam; Total Marks 41, Maximum marks 40, Time Limit 3 hours

December, 2023

1. Below a struct, a union and a type are defined. It is followed by a list of 10 function calls. **Write the output** when these function calls are made, by considering the function definitions that follow. Assume also that the appropriate `#include` statements exists. Note that in case of 1(j) there is a printf after the function returns. [10x1=10]

```
struct point {
   int x, y;
   struct point * next;
};
typedef struct point Point;
```

```
union id {
   char ids[100];
   int id_n;     // an int is 4 bytes
};
```

(a) `fun1a();`

(b) `fun1b();`

(c) `fun1c();`

(d) `fun1d();`

(e) `fun1e();`

(f) `fun1f();`

(g) `fun1g();`

(h) `fun1h();`

(i) `fun1i();`

(j) `printf("%d",fun1j(3));`

```
//------------ 1 (a) -----
void fun1a(){
  int a[10]={1,2,3};
  int *p = &a[1]; int *q=&a[2];
  int t = a[1]; a[1]=a[2]; a[2]=t;
  printf("%d %d",*p, *q);
}

//----------- 1 (b) -----
void fun1b() {
   int a[10]={1,2,3};
  int *p = &a[1]; int *q=&a[2];
  int *t = p; p=q; q=t;
   printf("%d %d",a[0],a[1]);
}

//---------- 1 (c) -----
void fun1c(){
   Point p={1,2,NULL}, q={3,4,NULL};
   p->next = &q;
   printf("%d %d",p->next->x,
        q.y);
}
```

```
// --------- 1 (d) --------------
void fun1d(){
   Point p={1,2,NULL}, q={3,4,NULL};
   p->next = &q;
   q = p;
   printf("%d",q->next->next->x);
}

//----------- 1 (e) -----
void fun1e(){
   Point *p=malloc(sizeof(Point));
   p->x=1;
   if( p->x == (*p).x ) printf("Yes");
      else printf("No");
}

// ---------- 1 (f) --------
void fun1f(){
   char a[10]="hello";
   char b[15]="bye";
   if ( a[5] == b[3] ) printf("YES");
     else printf("NO");
}
```

```
// ---------- 1 (g) -------
void fun1g(){
   char a[10]=
     {'\0','a','b','c','\0','d'};
   printf("%d",strlen(a+4));
}


// --------- 1 (h) -------------
void fun1h(){
    int a[5]={0,10,20,30,40};
    int i=4;
    do {
       printf("%d ",a[i]);
       i++;
    } while ( i < 4);
}
```

```
// --------- 1 (i) --------------
void fun1i(){
    int i=3, j=0;
    do {
        int i=0;
        j++;
        i++;
    } while ( j < 2 );
    printf("%d %d",i,j);
}
// --------- 1 (j) --------------
void fun1j(int n){  //note: question calls
                    //      this with n=3
    if ( n == 0 )
         return 0;
    return ( funj(n-1) * n);
}
```

2. **Write functions for the following** assuming the structure defined in Q1, and the provided `print()` function below. [4+4+2=10]

```
void print(struct point *head){
  Point * l = head;
  for( ; l!=NULL ; l=l->next)
     printf("%d %d\n", l->x, l->y);
}
```

   (a) **Write the recursive** function `void print_rev(Point *head);`
   This function prints the items of the list in reverse, ie the tail element is printed first and the head element is printed last. To do this **use the following** idea:
   1. if the list is empty i.e., head is a NULL pointer do nothing and return.
   2. Otherwise recursively call `print_rev` on the remainder of the list after the node pointed by head, then print the `x` and `y` value of the node pointed to by head.

   (b) **Write the function** `void delete_negatives(Point *head);`
   This function goes through the list and deletes and frees up every element in the list that has a negative `x` value; however, the function retains the first node pointed to by the header, independent of whether it's `x` is negative or not.

   (c) **Write the function** `Point * delete_all_negatives(Point * head);`
   This is like the previous function except that it also deletes the head node if it contains a negative `x` value. The function returns the head value of the resulting linked list after deletion.

3. **Write the function** `int merge(int a[], int n, int b[], int m, int c[]);`. [4]
   It merges two sorted arrays: `a[]` with `n` integers and `b[]` with `m` integers into a single sorted array `c[]`. You may assume that `c[]` has enough space.

4. **Write the function** `void merge_sort(int a[], int n);` [4]
   It sorts the given array `a[]` of `n` integers using the merge sort algorithm. Here is a summary of how you are expected to implement it:
   1. if n is 0 or 1 there is nothing to do, just return.
   2. Otherwise

(a) using `malloc()` create two smaller arrays a1[] and a2[] each capable of holding half the elements of a[]. Copy one half of a[] to a1[] and the other half to a2[].

(b) recursively call `merge_sort` to sort these two arrays a1[] and a2[] independently.

(c) Call the `merge()` function of the previous question to now merge `a1[]` and `a2[]` together into the original array `a[]`.

5. **Write the function** `voidinsertion_sort(int a[], in n);` Here is the basic idea : [5]

   (a) We note that initially just the element `a[0]` can be imagined to be a sorted array of one item.

   (b) if `a[0]...a[k]` is sorted then we can sort `a[0]..a[k+1]` by simply finding the correct index position for the value of `a[k+1]` (call that `v`) among `a[0]...a[k]`. If that index position is `p` , then we move all existing elements from `a[p]` to `a[k]` up by one position and then put `v` in `a[p]`.

6. This question is about arrays of structures. [2+4+2=8]

   (a) **Declare a structure** called `struct book`. It has two fields: an array of 10 characters called `title`. An integer called `npages`. Also **declare an array** of 20 such structures called `books[]` .

   (b) **Write a function** called `struct book * search(struct book a[], int n, char * x)` that takes an array of `n` books a[] and a character string `x`. It searches the array and returns the pointer to the unique structure in the array whose `title` matches the given string in `x`. If there are no structures with the `title` matching `x` or if there are multiple matches, then it returns the NULL pointer.

   (c) Assume the array `books[]` mentioned in part (a) exists and has 10 books in it already. Show how you will call `search()` to search for a book titled ”Forgotten” and use the returned value to print the number of pages in that book if it is unique.