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

Semester final Exam; Total Marks 41, Maximum marks 40, Time Limit 3 hours

November 20, 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. [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) `fun1j(3);`

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

//----------- 1 (b) -----
void fun1b() {
   char a[10]=
     {'\0','a','b','c','\0','d'};
   printf("%s %s %s",a,a+1,a+2);
}

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

// --------- 1 (d) -------------
void fun1d(){
    union id a,b;
    strcpy(a.ids,"hippopotamus");
    b.id_n=1;
    printf("%d %d", sizeof(a),sizeof(b));
}

//----------- 1 (e) -----
void fun1e(){
  Point p={0,0,NULL};
  Point q={300,400,NULL};
  sizeof(p)==sizeof(q) ?
  printf("Yes") : printf("No");
}

// ---------- 1 (f) --------
void fun1f(){
   int n=3, m=4;
   int *p=&n, *q=&m;
   int *t;
   t = p;  p = q;  q = t;
   printf("%d %d",n,m);
}

// ---------- 1 (g) -------
void fun1g(){
   char a[10]=
        {'\0','a','b','c','\0','d'};
    printf("%d",strlen(a+3));
}


// --------- 1 (h) -------------
void fun1h(){
    int a[5]={0,10,20,30,40};
    int i=2;    int *p=&a[i];
    while(i--)
       printf("%d ",*(p+i));
}
// --------- 1 (i) --------------
void fun1i(){
    int i=3, j=0;
    do {
        static int j = 13;
        j++;
        i++;
    } while ( i < 2 );
    printf("%d %d",i,j);
}
// --------- 1 (j) --------------
void fun1j(int x){
    switch(x-x){
    default:
        printf("1 ");
    case 2:
        printf("2 ");
        break;
    case 3:
        printf("3 ");
        break;
    }
}
```

2. **Write functions for the following** assuming the structure defined in Q1, and the provided `print()` function below. [3+3+3+3+2=14]

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

(*a*) `struct point * read();`
This function allocates a new `struct point`. It then reads `x` and `y` values from the user into the fields of this structure. It sets the `next` field to NULL. **Then it returns the pointer to this allocated structure**.

(*b*) `struct point * read_list();`
This function reads an integer n and then n points from the user by calling the `read()` function above to read and get a new `struct point` node. Each time a point is read it is added to the end of the linked list. **The head of the resultant linked list is returned.**

(*c*) `struct point find_xmax_point(struct point * head);`
For this function assume the list has at least one element in it and that all elements have distinct values for `x`. It finds the node with the maximum value of `x`. Then it **returns the value of that node** (notice that the return type is a structure **not a pointer** to a structure). The list is left unchanged. As an example, if this list has the values (1,1) (5,1) (4,5) in that order the return value is a structure with (5,1)

(*d*) `struct point * get_prev_xmax_ptr(struct point *head);`
For this function assume the same as for `find_max`. It is similar to `find_max`, however **it returns the pointer** to the node *previous to the node with the maximum* `x` value. If the maximum `x` is in the node pointed to by `head` it returns NULL. The list is left is unchanged. As an example, if this list has the values (1,1) (5,1) (4,5) in that order the return value is the pointer to the node in the list with value (1,1).

(*e*) Write a `main()` function. Declare a variable for the head of the linked list. Then call `read_list()` to read a list of items. Print the list by calling the provided `print()` function.

Call `find_xmax()`and print the returned point's `x` and `y` values. Lastly, call `get_prev_max()` and if the returned value is not NULL then print the returned `x` and `y` values of the node pointed to by the returned value.

3. **Write the function** `int search_sorted(int a[], int key, int i, int j);` . [4]
   It searches for the value `key` in the given increasing value sorted array `a[]` between the given indexes `i` and `j` . This function returns -1 if the value `key` is not found in `a[i], a[i+1],..,a[j]`. Otherwise it **returns the index** `k` such that `i` $\leq$`k` $\leq$ `j` and `a[k]` has the value `key`.

   You must use the **binary search**, method recapped below for you to expand on, and write the function:

   ```
   1. if  j  is smaller than  i  return -1 inidcating element not found.
   2. At the middle index of i and j, check if the key is found there:
       (a) if they key is found then return that index.
       (b) if not, search either the left or the right of that index, recursively.
   ```

4. For this question you need to sort an array using this recursive idea : [5]
   To sort items `a[0]...a[n-1]`, we first sort the items `a[1]...a[n-1]` using recursion. Then we bubble up `a[0]` to its appropriate position in `a[0]...a[n-1]`. This is done by a sequence of compare and swap of adjacent elements. **Complete the code below to implement the recursive function:**

   ```
   void sort(int a[], int n ){
      // Handle the recursion base case of n being at most 1; nothing to do!
      // Otherwise:
      //   1. call sort() recursively to sort the n-1 elements a[1]..a[n-1].
      //   2. In a loop bubble up a[0] until it reaches its correct position.
      //       i.e., repeated compare and swap  adjacent elements starting with a[0].
   }
   ```

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

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

   (b) **Write a function** called `greater()`that takes two structure **values** (not pointers) as parameters. It returns 1 if the `size` field of the first is greater than that of the second. It also returns 1 if the `size` fields are the same and the the `model` of the first appears after the the `model` of the second in dictionary order. Otherwise it returns 0. You may use the `strcmp` library function to compare two strings.

   (c) **Write a function** called `swap()` that takes the array `a[]` each of whose elements is a `struct pen` and two array indices i and j as parameters. It swaps `a[i]` with `a[j]`.

   (d) **Write a function** called `bubble_sort()`. It takes two parameters: the array `a[]` each of whose elements is a `struct pen` and an integer `n` indicating the number of elements in the array. It sorts the array using the bubble sort algorithm. For this you must use the `greater()` and `swap()` functions as described above. It returns nothing.