PROGRAMMING TECHNIQUES – 17/07/2023

The hard deadline to submit the self-evaluation report is 19/07/2023, at 23:59

NAME	
SURNAME	
MATRICOLA ID	

Theory section 1 (to be filled in the given paper)

EXERCISE 1 (5 points)

Given the following sequence of pairs, where relation i-j indicates that vertex i is adjacent to vertex j:

12-6, 2-3, 0-9, 5-4, 3-10, 10-8

Apply an on-line connectivity algorithm with quickfind. Nodes are named with integers in the range 0 … 12.

Questions & Answers:

- List as a sequence of integers the contents of array id after executing the step for pair 2-3
  id = 0 1 3 3 4 5 6 7 8 9 10 11 6

- List as a sequence of integers the contents of array id after executing the step for pair 5-4
  id = 9 1 3 3 4 4 6 7 8 9 10 11 6

- List as a sequence of integers the contents of array id after executing the step for pair 10-8
  id = 9 1 8 8 4 4 6 7 8 9 8 11 6

- Supposing that in the graph there are N vertices and that quickfind is used for the find operation, which of these notations expresses more precisely the worst case asymptotic complexity of the **union operation**?
  - O(N)
  - Θ(N) X
  - They are equivalent
  - None of them

  Brief explanation of the answer: when executing a union operation, the array is walked through to replace id[p] values with id[q] values. The cost is linear in the size of the array and, as the whole array is always walked through, it is Θ(N)

- Supposing that in the graph there are N vertices, that there are k pairs and that quickfind is used for the find operation, how many find operations will be performed by the **on-line connectivity algorithm**?
  - O(kN)
  - Θ(kN)
  - O(k)
  - O(2k)
  - Θ(k) X
  - Θ(2k)

  Brief explanation of the answer: given k pairs, 2k find operations are performed. The find operation has unit complexity. The number of find operations is thus Θ(k). Morover, as not all pairs require a union operation, whose complexity is linear in the size of the array, the complexity of the online connectivity algorithm with quickfind is O(kN).

EXERCISE 2 (5 points) TO BE FILLED IN THE GIVEN PAPER!!!

Sort the following integer array in ascending order using Shell sort:

25 3 12 37 4 82 5 0 19 10 17 45 43 42 51 44

- Which among the following sequences guarantees $O(N^{3/2})$ complexity?

  - ☐ 1 2 4 8 16 …
  - ☐ 1 2 3 4 6 8 9 12 …
  - X. 1 4 13 40 121 …
  - ☐ 1 5 19 41 109 209 …

- Given the sequence chosen at the previous point, given the size of the array to sort, what is the value h starts from?

  h = 13

- Briefly justify the value of h found in the previous point.

  Using Knuth's sequence, h=13 is the largest value of the sequence <= N (size of the array)

- Briefly explain what is an h-sorted array:

  If the subsequences containing items at distance h are sorted, the array is h-sorted.

- Once the necessary previous steps have been performed, show the 4-sorted array as a sequence of integers:

  4 3 5 0 19 10 12 37 25 42 17 44 43 82 51 45

- Is 3 4 82 5 19 0 17 45 44 a subsequence, a subarray, both a subsequence and a subarray, neither a subsequence nor a subarray of the original array?

  - ☐ Subsequence
  - ☐ Subarray
  - ☐ Both
  - X. Neither

  Briefly justify the answer: it is neither a subarray nor a subsequence as indices are not in increasing order (19 at index 8 precedes 0 at index 7).

NAME	
SURNAME	
MATRICOLA ID	

EXERCISE 3 (5 points) TO BE FILLED IN THE GIVEN PAPER!!!

Given the following piece of code:

```
#define R 2
#define C 4
struct person {
   char name[10];
   char surname[12];
   struct person *pPerson; }
int x, *p, mat[R][C], *m[R];
struct person s = {"Mary","Stuart",NULL}, *ps = &s;
```

Assuming a 64-bit architecture, and a 32-bit encoding for `int`, provide the storage size (**in bytes**) of the following expressions, briefly explaining your answer:

expression	*size (bytes)*	*BRIEFLY EXPLAIN WHY?*
`x`	4	Int is 32 bit → 4 bytes
`p`	8	p is a pointer. In a 64-bit architecture, size of a pointer is 8 bytes
`mat`	32	Matrix of 2x4 int elements. Size is 2*4*4 bytes
`mat[1]`	16	Row of the matrix: array of C int elements. Size is 4*4 bytes
`mat[1][1]`	4	Element of the matrix is an integer
`m[1]`	8	Element of index 1 of array m is a pointer to int → 8 bytes
`m`	16	Array of R pointers → size is R*size of a pointer → 16 bytes
`s.pPerson`	8	It is a pointer to struct person → size of a pointer is always 8 bytes
`ps`	8	It is a pointer to struct person
`ps->name`	10	It is an array of 10 char elements → size is 10*sizeof(char)

PROGRAMMING SECTION

EXERCISE 4 (4 points) TO BE FILLED IN THE GIVEN PAPER!!!

A program is executed from the Windows command line as follows: `C:\program.exe <option> <word>` where `<option>` can be one of the following: `-c, -C, -r` or `-R` and `<word>` is a string with no spaces. When `<option>` is `-c` (or `-C`), the program prints `<word>` on the screen with no repeated consecutive characters, by calling the function `printCompact`. When `<option>` is `-r` (or `-R`), it prints `<word>` in reverse, by calling the function `printReverse`. Fill the table below with the missing code in the corresponding boxes.

```
#include <stdio.h>
#include <string.h>
void printCompact( <1> word) {
  char ch; int i;
  for (i=0;i<strlen(word);i++)
    {
      if (i==0 || word[i]!=ch)
        putchar(word[i]);
      ch = <2>
    }
}
void printReverse( <3> word) {
  int i;
  for ( <4> )
    putchar(word[i]);
}
int main( <5> ) {
  if ( <6> ) {
    fprintf(stderr,"The number of the arguments is wrong!");
    return -1;
  }
  switch( <7> ) {
    case 'c': case 'C':
      printCompact(argv[2]);
      break;
    case 'r': case 'R':
      printReverse(argv[2]);
      break;
    <8>
      fprintf(stderr,"Invalid argument. Valid options are: -r,-R,-c,-C");
  }
  return 0;
}
```

BLOCK	MISSING CODE
<1>	char *
<2>	word[i];
<3>	char *
<4>	i = strlen(word)-1;i>=0;i--
<5>	int argc, char* argv[]
<6>	argc!=3
<7>	argv[1][1]
<8>	default:

SURNAME	
NAME	
MATRICOLA ID	

EXERCISE 5 (6 points) TO BE FILLED IN THE GIVEN PAPER!!!

In the following, you see three programs, each in two different versions (A and B). For each given version, you need to say whether it is syntactically and/or logically correct. If the answer is yes, write the corresponding output printed on the screen in the table below. Otherwise, briefly explain why the program is not correct.
NB. The two versions can be both correct, or both not correct!

1.A

```
#include <stdio.h>
int main(void) {
    char s[] = "1 2 3 10";
    int x,nc,sum = 0,cnt = 0;
    while (sscanf(s, "%d%n",&x,&nc)>0)
    {
        s=s+nc;
        sum+=x;
        cnt++;
    }
    printf("%.2f",(float)sum/cnt);

}
```

1.B

```
#include <stdio.h>
int main(void) {
    char s[] = "1 2 3 10";
    int x,nc,sum = 0,cnt = 0;
    char *s2 = s;
    while (sscanf(s2, "%d%n",&x,&nc)>0)
    {
        s2=s2+nc;
        sum+=x;
        cnt++;
    }
    printf("%.2f",(float)sum/cnt);

}
```

2.A

```
#include <stdio.h>
int main() {
    char s0[] = "abc";
    char s1[] = "abd";
    int i=0;
    while(s0[i]==s1[i] && s0[i]!='\0')
        i++;
    printf("%d",s0[i]-s1[i]);
}
```

2.B

```
#include <stdio.h>
#include <string.h>
 int main() {
    char s0[4] = "abc";
    char s1[4] = "abd";
    int i,flag = 1;
    for (i=0;flag && i<strlen(s0);)
        if (s0[i]!=s1[i])
            flag = 0;
        else
            i++;
    printf("%d",s0[i]-s1[i]);
}
```

3.A

```
#include <stdio.h>
int main() {
    char s[] = "mamma";
    char *p = s;
    int cnt = 0;
    while (*s!='\0')
        s++;
    printf("%d",s-p);
}
```

3.B

```
#include <stdio.h>
int main() {
    int cnt;
    char s[] = "mamma";
    for (cnt=0; s[cnt]!='\0';cnt++);
    printf("%d",cnt);
}
```

	CORRECT? (YES/NO)	IF YES, WRITE THE CORRESPONDING OUTPUT ON THE SCREEN. IF NO, BRIEFLY EXPLAIN WHY.
1.A	NO	s is an array, it cannot be re-assigned as if it was a pointer.
1.B	YES	4.00
2.A	YES	-1
2.B	YES	-1
3.A	NO	s is an array, it cannot be re-assigned as if it was a pointer.
3.B	YES	5

EXERCISE 6 (8 points) USE YOUR OWN PAPER!!!

In a C program, the following struct is used to represent a product in a warehouse:

```
typedef struct {char serialID[13]; int n_items;} product;
```

The `serialID` is a univocal identifier for a product, represented by an alphanumerical string of 12 characters without spaces. `n_items` represents the number of items that are available for the same product.

The list of products currently stored in the warehouse is contained in a text file, whose name is passed as argument from the command line. This file contains records of a maximum of 200 different products, reported in no specific order. Each line of the file reports the `serialID` and the number of items of a product, with spaces as separators. You can assume that the file is not empty, and that each product is reported only once in the file.

Consider the following fragment of code:

```
int main (int argc, char *argv[]) {
  product listProducts[200], *refProducts[200];
  int i, nprod = readProducts(listProducts, 200, argv[1]);
  for (i=0;i<nprod;i++)
     refProducts[i]=&listProducts[i];
  sortProducts(refProducts,nprod);
  // … rest of the program (omitted)
  return 0;
}
```

1) The function `readProducts` reads the input file and stores the data of the products into the array `listProducts`, returning the number of stored products to the caller.
2) The function `sortProducts` sorts the products in **alphabetical order by serialID,** by applying a **selection sort** algorithm. The function should not modify the original order of the array `listProducts`. Instead, it should sort an array of of **pointers to product** (see array `refProducts`).

The rest of the program is omitted.

You are required to implement the functions `readProducts` and `sortProducts`. The prototypes of the functions **must** be compatible with the corresponding function calls in the given code.

EXAMPLE OF CONTENT OF THE INPUT FILE

XY237H24ABCD 24

ZH117H24ABCD 2

AR217H24AACD 13

BB117H24AACD 1

```
int readProducts(product lProd[], int n, char* filename)
{
  FILE* fp = fopen(filename,"r");
  if (fp == NULL)
  {
    fprintf(stderr,"Error opening file %s", filename);
    exit(-1);
  }
  int np=0;
  while(np<n && fscanf(fp,"%s%d",lProd[np].serialID,&lProd[np].n_items)!=EOF)
    np++;
  fclose(fp);
   return np;
}

void sortProducts(product* refProd[],int np)
{
  int i,j,iMin;
  product* tmp;
  for (i=0;i<np-1;i++)
    {
      iMin = i;
      for (j=i+1;j<np;j++){
        if (strcmp(refProd[j]->serialID,refProd[iMin]->serialID)<0)
          iMin = j;
      }
      tmp = refProd[i];
      refProd[i] = refProd[iMin];
```

```
      refProd[iMin] = tmp;

    }
}
```