PROGRAMMING TECHNIQUES – 18/09/2023

The hard deadline to submit the self-evaluation report is 20/09/2023, at 23:59

SURNAME	
NAME	
MATRICOLA ID	

THEORY SECTION

EXERCISE 1 (5 points). - TO BE FILLED IN THE GIVEN PAPER!!!

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

12-5, 2-4, 0-2, 5-4, 9-11, 5-8

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

- Show arrays id and sz as sequences of integers after executing the step for pair 5-8
  id = 4 1 4 3 4 4 6 7 4 11 10 11 5
  sz = 1 1 1 1 6 2 1 1 1 1 1 2 1
- In an on-line connectivity algorithm with weighted quickunion, N being the number of vertices in the graph, the complexity of the find operation is:
  - X. O(logN)
  - ☐ Θ(logN)
  - ☐ O(N)
  - ☐ Θ(N)
  - ☐ O(1)
  - ☐ O(N)
  - ☐ O(NlogN)
  - ☐ O(N²)

  Briefly justify your answer: the find operation must in the worst case walk through a "chain" whose length is logarithmic in N. Thus T(N) = O(logN).

- Suppose, in an on-line connectivity algorithm with weighted quickunion, to turn the code in the left-hand side into the code in the right-hand side:

```
if (sz[i] <= sz[j]) {                    ⇒    if (sz[i] <= sz[j]) {
  id[i] = j; sz[j] += sz[i];                     id[j] = i; sz[i] += sz[j];
}                                              }
else {                                         else {
  id[j] = i; sz[i] += sz[j];                     id[i] = j; sz[j] += sz[i]
}                                              }
```

  - What change would be introduced? Would the resulting algorithm still be correct?
    Briefly justify your answer:
    the larger tree would be merged into the smaller one and not viceversa. The algorithm would be correct, as the principle of representing an element by means of the representative of its "chain" would be maintained.

  - Would the worst-case asymptotic complexity of the resulting algorithm change?
    Briefly justify your answer:
    yes, there wouldn't be any more guarantee of "chains" whose length is logarithmic in N.

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

Given the following function:

```
int f(int A[], int N, int x) {
  int i, j;
  for (i = 0; i < (N - 1); i++) {
    for (j = (i + 1); j < N; j++) {
      if (A[i] + A[j] == x) {
        return 1;
      }
    }
  }
  return 0;
}
```

Questions & answers:

- What is the worst-case asymptotic complexity of function f?
  - ☐ O(logN)
  - ☐ $\Omega(N)$
  - ☐ O(N)
  - X $O(N^2)$
  - ☐ $\Theta(N^2)$
  - ☐ $\Omega(N^2)$
- Briefly justify your answer:
  there are 2 nested loops, complexity is thus quadratic. As soon as the if condition is met, the function returns. Thus the number of iterations is at most quadratic: $T(N) = O(N^2)$
- Briefly explain what function f does on the array of integers A of size N:
  function f finds, if it exists, a pair of successive (not necessarily consecutive) elements whose sum is x. It returns 1 if such pair exists, 0 otherwise.
- Function f returns 0:
  - ☐ The first time the condition (A[i] + A[j] == x) is NOT met
  - ☐ Every time the condition (A[i] + A[j] == x) is NOT met
  - X If the condition (A[i] + A[j] == x) is NEVER met

  Briefly justify your answer:
  if no pair of consecutive elements whose sum equals x exists, the outer and the inner loops are executed without any early stop. When they are over, the return instruction returns 0.
- Function f returns 1:
  - X The first time the condition (A[i] + A[j] == x) is met
  - ☐ Every time the condition (A[i] + A[j] == x) is met
  - ☐ The last time the condition (A[i] + A[j] == x) is met

  Briefly justify your answer:
  as soon as the if condition is satisfied a return instruction is executed. It interrupts the loops and exits the function returning 1.

NAME	
SURNAME	
MATRICOLA ID	

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

Given the following piece of code:

```
#define N 12
typedef struct {
    char serialID[N];
    int n_items;
} product;
product p1 = {"abcd",0}, listProducts[N], *refProducts[N], *rP = &p1;
int *x = NULL, p[2][3];
float mat[4][3];
```

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

expression	*size (bytes)*	*BRIEFLY EXPLAIN WHY?*
`p1`	16	Struct with int (4 bytes) and string (12 bytes)
`rP->serialID[1]`	1	1 character, 1 byte
`p1.serialID`	12	String of size 12
`p`	24	2x3 matrix of int (size 4 bytes) -> total is 24 bytes
`listProducts[2]`	16	Type product: struct with int (4 bytes) and string (12 bytes)
`refProducts`	48	Array of pointers: N x 4 bytes = 48 bytes
`&p1`	4	Pointers are 4 bytes in a 32-bit architecture
`x`	4	Pointers are 4 bytes in a 32-bit architecture
`*x`	4	Integer has 32 bit encoding -> 4 bytes
`mat[1]`	12	Array of 3 float: 3xsizeof(float) = 12 bytes

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 <name_of_file>` where `<name_of_file>` is the name of a text file containing positive integer values, one per line. By calling the function `readValues`, the program 1) reads the values in the file and stores them in the array `vett`. If the file contains more values than the size of `vett`, such values are ignored; 2) computes the average value and prints it on the screen with two decimal digits. Complete the program by filling the table with the missing parts.

```
#include <stdio.h>
#define N 100

float readValues(char* filename, int values[], int nmax) {

  int nValues = 0;  // number of values read from the file
  int sum = 0;      // sum of the values read from the file

  FILE* fp = fopen(filename,"r");

  if (fp==NULL)
    return -1;

  while(nValues<nmax && fscanf(fp,"%d", <1> )!=EOF)
    {
      sum += <2>        // updates sum of values
      nValues ++;       // updates number of values
    }

  if (nValues>0)
    return( <3> );    // compute and return mean value
  else
    return -1;
}

int main(int argc, char* argv[]) {

  if ( <4> ) {
    printf("Number of arguments is wrong");
    return(-1);
  }

  int vett[N];   // array where the values should be stored
  float meanValue = readValues( <5> , <6> ,<7>);

  if (meanValue < 0)
    printf("File is empty or could not be opened");
  else
    printf("The mean value is: <8> );  // prints meanValue with 2 decimal digits

  // ... rest of the program (omitted)

  return 0;
}
```

BLOCK	MISSING CODE
<1>	&values[nValues]
<2>	values[nValues];
<3>	(float)sum/nValues
<4>	argc!=2
<5>	argv[1]
<6>	vett
<7>	N
<8>	%.2f", meanValue

SURNAME	
NAME	
MATRICOLA ID	

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

Given the following functions:

```
void func1(int mat[][2],int n)
{
  int i,j;
  for(i=0; i<n; i++) {
    for(j=0; j<n; j++)
       printf("%3d",*(*(mat+j)+i));
    printf("\n");
  }
}
```

```
void func2(int mat1[][2],int mat2[][2],int n){
  int i,j,k,r;
  for (i=0;i<n;i++) {
    for (j=0;j<n;j++) {
        r=0;
        for (k=0;k<n;k++)
          r += *(*(mat1+i)+k) * *(*(mat2+k)+j);
        printf("%3d",r);
      }
      printf("\n");
    }
}
```

```
void func3(int mat[][2],int n)
{
  int k,s=0;
  for (k=0;k<n;k++)
      s+=*(mat[k]+k);
  printf("%3d",s);
}
```

```
void func4(int mat[][2],int n)
{
  int k,s=0;
  for (k=0;k<n;k++)
      s+=*(mat[k]+n-k-1);
  printf("%3d",s);
}
```

1) Assuming that the following variables have been declared and initialized as follows:

```
int a[2][2]={{1,2},{3,4}};
int b[2][2]={{1,0},{0,1}};
int c[3][3]={{1,1,2},{1,2,2},{1,1,2}};
```

write the output on the screen corresponding to each of the given function calls. **If no output is produced, you are required to write N/A**:

FUNCTION CALL	**OUTPUT ON SCREEN**	FUNCTION CALL	**OUTPUT ON SCREEN**
`func1(a,2);`	1 3 2 4	`func2(a,c,2);`	5 3 11 7
`func1(c,2);`	1 2 1 1	`func3(a,2);`	5
`func2(a,b,2);`	1 2 3 4	`func3(c,2);`	2
`func2(a,a,2);`	7 10 15 22	`func4(c,2);`	3

1) re-write the instructions in bold in func1,func2, func3 and func4, using the 2-d array notation instead of the pointer notation (NB: the function behaviour must be the same!):

ORIGINAL CODE (POINTER NOTATION)	CORRESPONDING CODE (2-D ARRAY NOTATION)
`printf("%3d",*(*(mat+j)+i));`	`printf("%3d",mat[j][i]);`
`r += *(*(mat1+i)+k) * *(*(mat2+k)+j);`	`r += mat[i][k] * mat[k][j];`
`s += *(mat[k]+k);`	`s += mat[k][k];`
`s += *(*(mat+k)+n-k-1);`	`s += mat[k][n-k-1];`

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[15]; int n_items;} product;
```

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

The list of products currently stored in the warehouse is contained in a text file, whose name is passed as the first argument from the command line. This file contains records of a maximum of 500 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. The file may contain multiple instances of the same product, or no products at all.

Consider the following fragment of program:

```
int main (int argc, char *argv[]) {
  product listProducts[500], *refProducts[500];
  int nprod = readProducts(listProducts, 500, argv[1]);
  int nsel = selectProducts(listProducts, nprod, refProducts, argv[2]);
  // … rest of the program (omitted)
  return 0;
}
```

1) The function `readProducts` reads the input file and stores the data of the products in the array `listProducts`, returning the number of stored products to the caller. In case the same product appears more than once in the file, it should be stored only once in the array, and the corresponding number of items should be summed to the pre-existing record.
2) The function `selectProducts` selects all the products of `listProducts` whose number of items exceeds a certain threshold, that is a positive integer value given as second argument from the command line when executing the program. The function should store the pointers to the selected products in the array `refProducts`, and return the total number of selected products to the caller.

The rest of the program is omitted.

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

EXAMPLE OF CONTENT OF THE INPUT FILE input.txt:

```
XY237H24ABCDQQ  24
ZH117H24ABCDQQ  2
AR217H24AACDQQ  13
BB117H24AACDQQ  1
ZH117H24ABCDQQ  2
```

Supposing to execute the program from the Windows command line as `C:\progr.exe input.txt 3`, the selected products should be XY237H24ABCDQQ, ZH117H24ABCDQQ and AR217H24AACDQQ.

PROPOSED SOLUTION

```
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); }
  char serialID[15];
  int i,n_items,np=0;
  while( fscanf(fp,"%s%d",serialID,&n_items)!=EOF)
  {
       for(i=0;i<np && strcmp(serialID,lProd[i].serialID)!=0;i++);
       if (i!=np)
           lProd[i].n_items += n_items;
       else
       {
           if (np<n)
           {
               strcpy(lProd[np].serialID,serialID); lProd[np].n_items = n_items;
               np++;
            }
        }
  }
  fclose(fp);
return np;
}
```

```
int selectProducts(product lProd[],int np,product* refProd[],char* threshold)

{

    int i, ns = 0;

    for (i=0;i<np;i++)

        if (lProd[i].n_items > atoi(threshold))

            refProd[ns++] = &lProd[i];

    return ns;

}
```