PROGRAMMING TECHNIQUES – 23/06/2022

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, 5-2, 0-9, 9-3, 12-9, 6-9

apply an on-line connectivity algorithm with quickunion. Nodes are named with integers between 0 and 12.

Questions & Answers:

- What is the complexity of the union and of the find operations with quickunion?

  union: find:

- In the Online connectivity algorithm is an explicit representation of the graph needed? Justify your answer.

- What is the size of array id? Justify your answer.

- List the contents of array id as a sequence of integers
  - after step 1 (steps start from 1)

    id:

  - after step 3

    id:

  - after step 5

    id:

- What represents node 6 at the end of the algorithm?

Theory section 2 (to be filled in the given paper)

EXERCISE 2 (5 points)

Sort the following integer array in ascending order by counting sort:

$12_1\ 5_1\ 4_1\ -1\ 4_2\ 9_1\ 6\ 12_2\ 4_3\ 9_2\ 3\ 9_3\ 5_2$

Subscripts identify instances of the same key.

Questions & Answers:

- What is the range of the data in this instance of the problem?

- What is the value of k in this instance of the problem?

  k =

  Short explanation:

- List as a sequence of integers the content of array C of simple occurrences

  C:

- List as a sequence of integers the content of array C of multiple occurrences

  C:

- At what index in the result array B will key 6_2 be stored?

NAME	
SURNAME	
MATRICOLA ID	

Theory section 3 (to be filled in the given paper)

EXERCISE 3 (5 points)

Given the following piece of code:

```
#define NR 4
#define NC 5

struct student {
char name[10]; char surname[12]; char id[6]; struct student *pStud;
};
float x, *p, mat[NR][NC], *m[NR];
struct student s = {"Paolo","Rossi","12345", NULL}, *ps = &s;
```

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

expression	*size (bytes)*	*BRIEFLY SAY WHY?*
`x`		
`p`		
`mat`		
`mat[0]`		
`mat[0][NR-1]`		
`m`		
`m[2]`		
`s.name`		
`s.pStud`		
`ps`		
`ps->name[3]`		
`(*ps).surname`		

Programming section (use your own papers)

EXERCISE 4 (4 points)

Write a function with prototype `void reverseString(char* s);` that reverses the string pointed by s. For example, the string "ciao" should become "oaic", "xY21" should become "12Yx", etc.
NB: you are required to implement the function from scratch, without using `strrev()`

EXERCISE 5 (6 points)

A text file contains a list of integer values in a 0 to 99 range, separated by either spaces or newline characters. Write a C function with prototype `int printTens(char* namefile);` that receives the name of the input file as the argument and prints on the screen the occurrence of values belonging to each range of tens (range 1: 0 to 9, range 2: 10 to 19, range 3: 20 to 29… range 10: 90 to 99). The function should return to the caller a 1 to 10 value, corresponding to the range with highest occurrence of values (in case of ranges with equal occurrence of values, choose the lowest one).

Example:

Content of the file: `0 58 21 50 21 53 7 22 24 51`
Output of the function:

```
2 value/s in range 1
4 value/s in range 3
4 value/s in range 6
```

Value returned by the function: `3`

EXERCISE 6 (8 points)

In a C program, the following struct is used to represent a schoolkid of a primary school:

```
typedef struct {char name[30]; char surname[30]; int age;} kid;
```

A text file, whose name is passed as first argument from command line, contains records of a maximum of 100 kids. Each line reports the name, surname, and age of a kid, with spaces as separators. Names and surnames do not contain any spaces.

Consider the following piece of code:

```
int main (int argc, char *argv[]) {
  kid listKids[100], *selKids[100];
  int ns,nsel;
  ns = readKids(listKids, 100, argv[1]);
  nsel = selectKids(listKids, ns, selKids, atoi(argv[2]));
  printKids(selKids, nsel);
  … // rest of the program (omitted)
  return 0;
}
```

1) The function `readKids` (omitted) reads the file and stores the data of the kids into the array `listKids`, returning the number of read kids to the caller.
2) The function `selectKids` selects the kids whose age is lower than a given threshold, which is the second argument passed from the command line. The pointers to the selected kids are stored by the function in the array `selKids`, and the total number of selected kids is returned to the caller.
3) The function `printKids` prints the list of selected kids on the screen, one kid per line, with the same format as in the input file.

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

#include <stdio.h>

FILE *fopen(char *filename, char * mode) – Opening a file (mode: "r" reading – "w" writing – "a" append)
FILE *freopen(char *filename, char * mode, FILE *file_pointer) – Reassign a file pointer to a different file
int fclose(FILE *file_pointer) – Closing a file
int feof(FILE *file_pointer) – Checks if end-of-file has been reached.
int fflush(FILE *file_pointer) - Empties file's buffer.
int getchar(void) – Reads a character from "stdin" (keyboard)
int fgetc(FILE *file_pointer) – Gets a character from a file
char *gets(char *buffer) - Reads a line from "stdin" (keyboard)
char *fgets(char *string, int maxchar, FILE *file_pointer) - Reads a line from a file
int printf(char *format _string, …) – Writes formatted output on "stdout" (screen)
int fprintf(FILE *file_pointer, char *format_string, …) – Writes formatted output on a file.
int sprintf(char *string, char *format_string, …) – Writes formatted output on a string.
int fputc(int c, FILE *file_pointer) – Writes a character on a file.
int putchar(int c) – Writes a character on "stdout" (screen).
int puts(char *string) - Writes a string on "stdout" (screen).
int fputs(char *string, FILE *file_pointer) - Writes a string on a file.
int scanf(char *format_string, args) – Reads formatted input from "stdin" (keyboard)
int fscanf(FILE *file_pointer, char *format string, args) – Reads formatted input from a file.
int sscanf(char *buffer, char *format_string, args) – Reads formatted input from a string.
EOF – end of file (costant with negative value)
NULL - null pointer (value 0)

#include <stdlib.h>

double atof(char *string) – Converts a string into a floating point value.
int atoi(char *string) – Converts a string into an integer value.
int atol(char *string) – Converts a string into a long integer value.
void exit(int val) – Terminates the program returning the value 'val'.
EXIT_FAILURE – constant highlighting the unsuccessful termination of the program with exit(); non zero value.
EXIT_SUCCESS - constant highlighting the successful termination of the program with exit(); zero value.

#include <string.h>

char *strcpy(char *s1, char *s2) - Copies s2 in s1. Returns s1
char *strncpy(char *s1, char *s2, size_t n) – Copies the first "n" characters of s2 in s1. Returns s1.
int strcmp(char *s1, char *s2) - Compares s1 and s2 to determine the alphabetical order (<0, s1 precedes s2, 0 equal, >0 s1 follows s2)
int strncmp(char *s1, char *s2, size_t n) – Compares the first "n" characters of two strings.
int strlen(char *string) – Determines the length of a string.
char *strcat(char *s1, char *s2, size_t n) - Links s2 to s1. Returns s1
char *strncat(char *s1, char *s2, size_t n) - Links "n" characters of s2 to s1. Returns s1
char *strchr(char *string, int c) – Finds the first occurrence of the character 'c' in string; returns a pointer to the first occurrence of c in s, NULL if not present.
char *strrchr(char *string, int c) – Finds the last occurrence of the character 'c' in string.
char* strstr(char* s, char* t) – Returns a pointer to the first occurrence of t in s. returns NULL if not present.
char* strtok(char* s, const char* t) – Decomposes s in tokens, the characters that limit the tokens are contained in t. returns the pointer to the token (NULL if any is found). At the first call the string to be decomposed is s and the characters delimiting the various tokens in t. To operate on the same string, at following calls NULL has to be passed instead of s.

#include <ctype.h>

int isalnum(int c) – True if 'c' is alphanumeric.
int isalpha(int c) – True if 'c' is an alphabetic charater.
int iscntrl(int c) – True if 'c' is a control character.
int isdigit(int c) - True if 'c' is a decimal digit.
int islower(int c) - True if 'c' is lowercase.
int isprint(int c) - True if 'c' is a printable character.
int ispunct (int c) - True if 'c' is a punctuation character.
int isspace(int c) - True if 'c' is a space character.
int isupper(int c) - True if 'c' is uppercase.
tolower(int c) – Converts 'c' to lowercase.
int toupper(int c) – Convert 'c' to uppercase.

#include <math.h>

int abs (int n) – integer absolute value
long labs(long n) – long absolute value
double fabs (double x) – absolute value of x
double acos(double x) - arccosine
double asin(double x) - arcsin
double atan(double x) - artangent
double atan2(double y, double x) – arctangent of y/x.
double ceil(double x) – round up value of x.
double floor(double x) – round down value of x.
double cos(double x) – cos (x in radians)
double sin(double x) – sin (x in radians)
double tan(double x) – tan (x in radians)
double cosh(double x) – hyperbolic cosine
double sinh(double x) – hyperbolic sin
double tanh(double x) – hyperbolic tangent
double exp(double x) - e^x
double log(double x) - log(x).
double log10 (double x) – logarithm base 10
double pow (double x, double y) - x^y
int rand (void) – random integer between 0 and RND_MAX.
int random(int max_num) – random value between 0 and max_num.
void srand(unsigned seed) – initialize the sequence of random values
double sqrt(double x) – square root

#include <limits.h>

INT_MAX – Maximum value that can be represented by int variable.
INT_MIN – Minimum value that can be represented by int variable.
LONG_MAX - Maximum value that can be represented by long variable.
LONG_MIN - Minimum value that can be represented by long variable.

#include <float.h>

FLT_MAX, DBL_MAX - Maximum value that can be represented by float (or double) variable.
FLT_MIN, DBL_MIN - Minimum value that can be represented by float (or double) variable.