uc3m | Universidad Carlos III de Madrid
Departamento de Ingeniería Telemática

SYSTEMS ARCHITECTURE **November 9th 2022**

SURNAME: ....
NAME: ....

Complete exam **Time: 100 min**

General instructions:

It is forbidden to use additional material on paper or other devices, the use of headphones, communication with other people, leaving the seat during the exam, and the use of additional software.

TEST (3 points + 0.6 extra points, 20 minutes)

Instructions:

There is only one correct answer for each question. Incorrect answers do not add or subtract points and correct answers add 0.3 points.

1. What is the output of the following code? Note that A=65 and a=97.

```
#include <stdio.h>

int caesar(char letter){
      if(letter >= 'A' && letter <= 'Z'){
            return 'A' + (letter-'A'+3)%26;
      }
      else if(letter >= 'a' && letter <= 'z'){
            return 'a' + (letter-'a'+3)%26;
      } else {
            return -1;
      }
}

int main(int argc, char **argv){
      char input[3] = "Yes";
      for(int i=0; i<3;i++){
            input[i] = caesar(input[i]);
            printf("%c", input[i]);
      }
      printf("\n");
      return 0;
}
```

a) Yes
b) 66104118
c) Bhv
d) The code produces a compilation error

2. Given the following data definitions and declarations, we want to initialize the field "id" of "a" and "b" with the values 50 and 60, respectively. Select the correct option.

```
struct student{
  int id;
  unsigned char name[40];
  int *ptr_int;
  struct student *next;
};

struct student a, b, *ptr_a;
ptr_a = &a;
ptr_a-> next = &b;
```

a) (*ptr_a).id =50; (*ptr_a).next.id = 60;
b) (*ptr_a).id =50; (*ptr_a).next->id = 60;
c) ptr_a.id =50; ptr_a.next.id = 60;
d) ptr_a->id = 60; ptr_a->next->id = 50;

3. How many lines are shown by default when using the “l” option in the gdb debugger if the program is not executing yet?
a) 5
b) 10
c) 15
d) The number depends on the number of lines of the main function

4. How many allocs and frees does Valgrind show when executing this code as “*./q3a aa bb cc*”?

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

struct node{
      char *name;
      struct node *next;
};


int main(int argc, char **argv){
      struct node *head = NULL;
      for(int i=1;i<argc;i++){
            char *name = strdup(argv[i]);
            struct node *new_node = (struct node *) malloc(sizeof(struct
node));
            new_node->name = name;
            new_node->next = head;
            head = new_node;
      }
      struct node *tmp = head;
      for(int i=1;i<argc;i++){
            printf("%s\n", tmp->name);
            tmp=tmp->next;
      }
      struct node *aux = head;
      for(int i=1;i<argc;i++){
            head = head->next;
            free(aux->name);
            free(aux);
            aux = head;
      }
      return 0;
}
```

a) 3 allocs, 3 frees
b) 4 allocs, 4 frees
c) 7 allocs, 7 frees
d) 9 allocs, 6 frees

5. Given the following implementation of a dynamic array. We want to delete the element index, with *index < num_ocuppied -1 and num_occupied > 1*. Choose the correct piece of code:

```
struct item{
   int id;
   long dato;
};
struct collection{
   int total;
   int num_occupied;
   struct item *array;
};
```

a) num_occupied--;
   memcpy(&(my_col[index]), &(my_col[num_occupied]), sizeof(struct item));
b) memcpy(&(my_col[index]), &(my_col[num_occupied]), sizeof(struct item));
   num_occupied--;
c) num_occupied--;
   strcpy(my_col[index], my_col[num_occupied]);
d) free(my_col[index].dato);
   num_occupied--;
   memcpy(&my_col[index], &my_col[num_occupied], sizeof(struct item));

6. What is the output of this code if “chmod 333 file.txt” is used before executing this code?

```
#include <unistd.h>
#include <stdio.h>

int main(){
      if (access("file.txt", R_OK) == 0){
            printf("r");
      } else {
            printf("-");
      }
      if (access("file.txt", W_OK) == 0){
            printf("w");
      } else {
            printf("-");
      }
      if (access("file.txt", X_OK) == 0){
            printf("x");
      } else {
            printf("-");
      }
      printf(" file.txt\n");
}
```

a) -wx file.txt
b) r-x file.txt
c) rw- file.txt
d) rwx file.txt

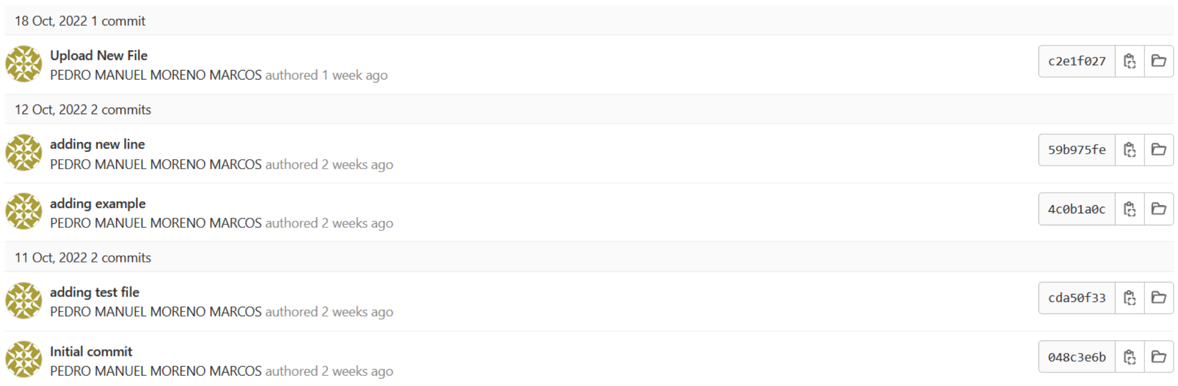
7. A hash function is:
a) A function that, given a key, returns the table index of the hash table where the element is stored (or where you must store it).
b) A function that, given a key, returns the element stored in the hash table.
c) A function that, given an element, returns its key in the hash table.
d) A function that, given a key, returns the index of the linked list where the element is stored inside the hash table

8. What is the output after executing this function?

```
#include <stdarg.h>
#include <stdlib.h>
#include <stdio.h>

void function(int num,...){

  va_list arg;
  int i;
  va_start(arg, num);

  for (i=0; i< num; i++){
        printf("%d",atoi(va_arg(arg, char *)));
  }
  printf("\n");
  va_end(arg);
}

int main (){
  function(2,"5","6","7");
  return 0;
}
```

a) 567
b) 18
c) 56
d) There is an error because of the input values of atoi

9. Given the following screenshoot, what were the possible commands the developer used the second time he did a commit in this project?



18 Oct, 2022 1 commit

Upload New File
PEDRO MANUEL MORENO MARCOS authored 1 week ago — c2e1f027

12 Oct, 2022 2 commits

adding new line
PEDRO MANUEL MORENO MARCOS authored 2 weeks ago — 59b975fe

adding example
PEDRO MANUEL MORENO MARCOS authored 2 weeks ago — 4c0b1a0c

11 Oct, 2022 2 commits

adding test file
PEDRO MANUEL MORENO MARCOS authored 2 weeks ago — cda50f33

Initial commit
PEDRO MANUEL MORENO MARCOS authored 2 weeks ago — 048c3e6b

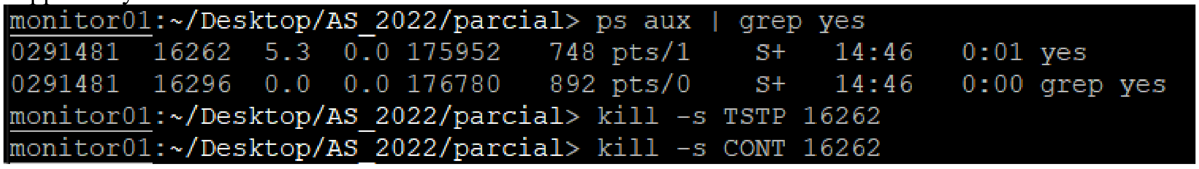
a) (1) git add ., (2) git commit “adding new line”, and (3) git push
b) (1) git add ., (2) git commit -m “adding new line” and (3) git push
c) (1) git add ., (2) git commit -m “adding test file” and (3) git push
d) (1) git commit -m “adding new line”, (2) git add ., and (3) git push

10. When executing the following code:

```
#include <stdio.h>
#include <stdlib.h>
#include <sys/types.h>
#include <sys/wait.h>
#include <unistd.h>
#include <errno.h>
#include <string.h>

int main(){
   pid_t  pid, pid_fin;
   pid = 0;
   int status;
   printf("Line A\n");
   pid = fork();
   printf("Line B\n");
   int myvar = 2;

   if (pid < 0){
      exit(EXIT_FAILURE);
   } else if (pid == 0){
       myvar = 5;
     _exit(EXIT_SUCCESS);
   } else{
       system("date");
     pid_fin = wait(&status);
     if (pid_fin == -1)
       perror(strerror(errno));
     else{
       printf("Value: %d\n", myvar);
     }
   }
   exit(EXIT_SUCCESS);
}
```

a) Value: 2 is printed once
b) The child process replaces its memory space with the program to print the date
c) Lines A and B are displayed twice
d) All the other options are correct

11. If you execute process "yes" and execute "kill -s TSTP 16262" afterwards as shown in the picture. What happens if you then execute "kill -s CONT 16262"



```
monitor01:~/Desktop/AS_2022/parcial> ps aux | grep yes
0291481  16262  5.3  0.0 175952   748 pts/1    S+   14:46   0:01 yes
0291481  16296  0.0  0.0 176780   892 pts/0    S+   14:46   0:00 grep yes
monitor01:~/Desktop/AS_2022/parcial> kill -s TSTP 16262
monitor01:~/Desktop/AS_2022/parcial> kill -s CONT 16262
```

a) The process will continue executing as it was previously stopped with TSTP
b) There will be an error because the process was terminated and cannot be continued
c) There will be an error because the process was killed and cannot be continued
d) There will be an error because the process was interrupted and cannot be continued

12. When reading from a pipe with the function read(), when the operating system returns a value equal to 0?
a) When there is no data to be read.
b) When the timer set by the alarm() function raises the signal alarm without receiving any data.
c) When there are no open writing descriptors associated to the pipe.
d) Never. It always returns the number of bytes read.

PROBLEMS (7 points, 80 minutes)
PROBLEM 1 (3.5 points)

Previous considerations:
1: Problem 1 has 3 sections.
2: Declare and use the variables you estimate necessary. Give then proper initial values to avoid errors.
3: Making unnecessary initializations is not penalized, however NOT doing necessary initializations DOES reduce the grade.

Part 1.1 Variable Parameters (qualifies 5 points over 10)
(This is needed for 1.2, not for 1.3)
Write a main program that receives 1 or 2 parameters of type int. The first parameter is mandatory, <P_AGE> integer and must be an age. The second is optional, and must be a <P_NUM> integer containing the number of people we are going to use. If not included, the default value is considered to be 3.

- Part A: If you include both parameters, it prints the message "Three arguments. There are <P_NUM> students with <P_AGE> years". Then a function from the second paragraph must be called. The function header is:
  void part_a(int p_age, int p_num);
- Part B: If only <P_AGE> is included, it prints the message "Two arguments. There are 3 students with <P_AGE> years". Nothing else is done.
- If there is another number of parameters, it prints the message "Incorrect number of parameters".

Examples of execution:
./partA 1
Two arguments. There are 3 students with 1 years.
./partA 19
Two arguments. There are 3 students with 19 years.
./partA 19 1
Three arguments. There are 1 students with 19 years.
./partA 20 4
Three arguments. There are 4 students with 20 years.

Part 1.2 Structure and getline (qualifies 2.5 points over 10)
(Depends on 1.1 but can be done without 1.1)

- Defines outside the main function the type of the Student structure.
  Name field of type string
  Age field of type integer

- Defines a function that prints the id, name and age. For clarity, insert a tab to the left of all fields.
  *void print_struct(int student_id, struct Student *p_student){...}*

Example of the first student:
  Student id: 1
  Name: <name>
  Age: <age>

- Define a function for the main of the first section.
  void part_a(int p_age, int p_num){...}

For each student:
1. Prints *"Enter the name: "*. Uses getLine(&line, &len, stdin) to ask for a person's name.
2. Creates an element named p_student of type Student. Assigns the string value received from getLine to the name field. Assigns the integer value of the <P_AGE> parameter to the age field. It must always be the same value.
3. Print the student_id and fields of the p_student element on the screen. The print_struct function defined above must be called. The student_id is an integer and corresponds to 1 for the first student, 2 for the second, etc.

It goes back to point one until there are no more students left.

Examples of execution:
./partA 20 4
Three arguments. There are 4 students with 20 years.

```
Enter the name: Alejandro
        Student id: 1
        Name: Alejandro
        Age: 20
Enter the name: Lucia
        Student id: 2
        Name: Lucia
        Age: 20
Enter the name: Marta
        Student id: 3
        Name: Marta
        Age: 20
Enter the name: Alvaro
        Student id: 4
        Name: Alvaro
        Age: 20
```

Part 1.3 Dynamic Integer Array (qualifies 2.5 point over 10)
(Independent of the above)

- Create a function called print_array that receives as parameter an array of integers and its size. It is called twice in main to print {20, 21, 23} and {20, 21, 23, 3, 4, 5}.

```
void print_array(int *array, int size){
        printf("{");
        for (int i=0; i< size-1; i++)
                printf("%i, ", array[i]);
                printf("%i}\n", array[size-1]);
}
```

Example of printing integer array:

```
{20, 21, 23}
{20, 21, 23, 3, 4, 5}
```

In the main:

- An array of integers of size 3 is to be allocated memory, filled, and printed.
    - Create an array of type integer called first_values.
    - Initialize it to NULL and dynamically allocate the memory indicated by the variable size. To do this, create an integer variable size = 3.
    - The first element of the array must be 20. The second adds 1 to the initial value. The third adds 2 to the previous value: {20, 21, 23}... and so on up to size.
- You must reallocate the memory to size 6, finish filling the array and print it.
    - Create another integer array called second_values.
    - Initialize it to NULL and dynamically reallocate the memory from 3 to 6. To do this, create another integer variable called new_size = 6.
    - Once the size of second_values is defined, make first_values = second_values so that first_values has the new size 6. Fill the rest of the array values with the index value with the position. It should be {20,21,23,3,4,5}.

Note: You have to change the size of an array. When you use **realloc** never assign directly the pointer that returns to the old one, because it can be NULL what the function returns. Always use an auxiliary variable like **second_values** to **realloc**, check that it is NOT NULL and then assign its value to the old pointer **first_values**.

Examples of execution:

```
./ex3a
{20, 21, 23}
{20, 21, 23, 3, 4, 5}
```

PROBLEM 2 (3.5 points)

Previous considerations:
1: The ordered steps suggested bellow for the problem's resolution are mere indications, they are NOT requirements as there can be many ways of solving the problem. (Though every code written in ***C*** language must be correctly sequentially ordered, of course).
2: Declare and use the variables you estimate necessary. Give then proper initial values to avoid errors.
3: Making unnecessary initializations is not penalized, however NOT doing necessary initializations DOES reduce the grade.
4: In the appendix at the end of the problem (see below) there is a list of standard library function declarations aimed to help in resolving this problem. Each item in the list consists of the first lines of information retrieved with the standard Unix *man* command. You can use this information to know which *#include* files are needed before using the functions. *Not all functions in the list are necessary*.

Part 1 (qualifies 2 points over 10)
You must build a function called ***get_number_of_bytes,*** that yields the correct number of chars (bytes) contained in a text file whose name is given to the function as a parameter. For this you shall:

- (0,3 points) Write the first implementation line of the function, provided that it must return in integer; that the name of the function must be *get_number_of_bytes* and that it receives as its only parameter a standard C language text string (ended with '\0') containing the name of the file to process.
- (0,2 points) Write the code needed to open the file in *read* mode, whose name is in the string given as parameter to the function.
- (0,1 points) Manage the possible error that can be returned when the opening fails. Return a negative value in this case.
- (1 point) Implement a loop that counts the number of characters while the opened file is being readed.
- (0,2 points) Release the resources associated to the use of the file (i.e., shall close it).
- (0,2 points) Return the byte count as result if there were no errors. Otherwise, a negative integer if any error occurred.

Part 2 (qualifies 7 points over 10)
Write the ***main*** function of a multiprocessing program that receives as input arguments the names of an undetermined amount of text files containing plain text. This program also …

- Shall print the number of characters (bytes) contained in each input file.
- Shall create a child process per file name given. This child process will process the file whose name is provided.
- The parent process will release all the assigned resources related to child process management. This means that there cannot remain *zombie* children processes after parent's execution.

To write this ***main*** function you must:

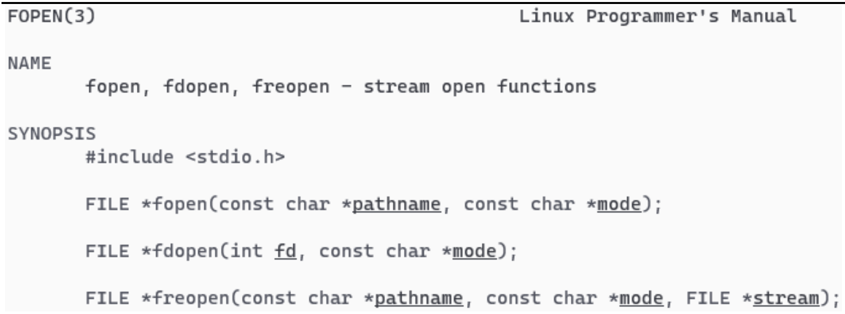
- (0,1 puntos) Write the first implementation line of ***main***, which is the standard one used along this course.
- (0,1 puntos) Ensure the program returns successfully, not doing anything whether it receives no parameters.
- (4 puntos) Write the ***for*** loop that creates one child process per input file name introduced in the command line and delivered through main's arguments.
- (0,3 puntos) Notify, printing in console, the failure when any child process creation cannot be performed. This fact will NOT stop the program, nor any other part of it.

Moreover:

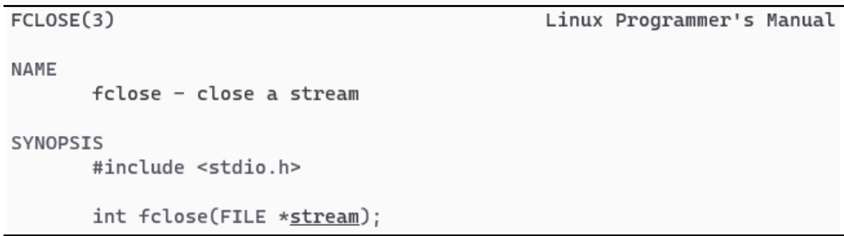
- (0,1 puntos) Any given child process will get the number of bytes in a file by calling the *get_number_of_bytes* function. This function and its interface will be described in the first part of the problem.
- (0,2 puntos) Any given child process shall print, in the console, the number of bytes in the file computed by *get_number_of_bytes*. The process must notify, through the console, whether the computation returned an error value.
- (0,1 puntos) Each of the children shall end after printing its results. At the same time, it shall return a constant suited to whether the process work was successful or not.
- (2 puntos) Just before the end, the parent process shall wait for the conclusion of all their children. It shall free all resources obtained (transparently) for their execution.
- (0,1 puntos) The parent process will exit the main function returning a value meaning success, in all cases.

Part 3 (qualifies 1 point over 10)
Modify the ***main*** function above, in such a way that there cannot be more than three child processes running at a given time. That means that the parent process must *wait* when one third process is launched, till one of the three ends its execution.
***Suggestion*:** count the number of correctly launched child processes. *wait* when the count reaches three and decrease the counter when the wait finishes.

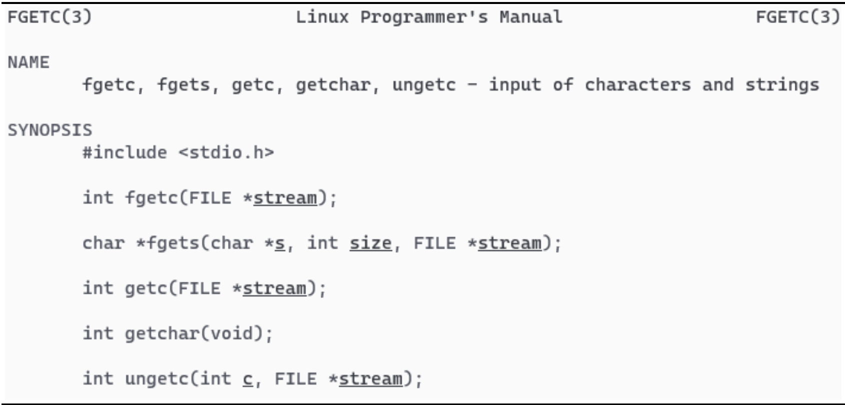
Appendix: Some standard C library functions documentation retrieved with the ***man*** Unix command.



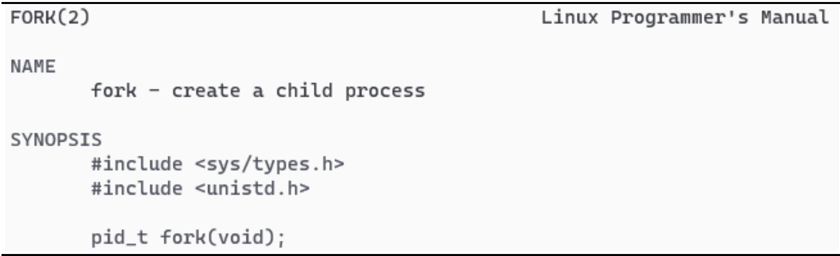
```
FOPEN(3)                                   Linux Programmer's Manual

NAME
       fopen, fdopen, freopen - stream open functions

SYNOPSIS
       #include <stdio.h>

       FILE *fopen(const char *pathname, const char *mode);

       FILE *fdopen(int fd, const char *mode);

       FILE *freopen(const char *pathname, const char *mode, FILE *stream);
```



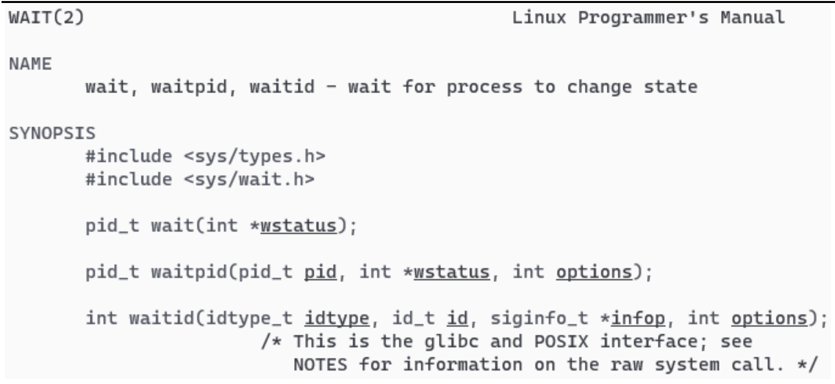
```
FCLOSE(3)                                  Linux Programmer's Manual

NAME
       fclose - close a stream

SYNOPSIS
       #include <stdio.h>

       int fclose(FILE *stream);
```



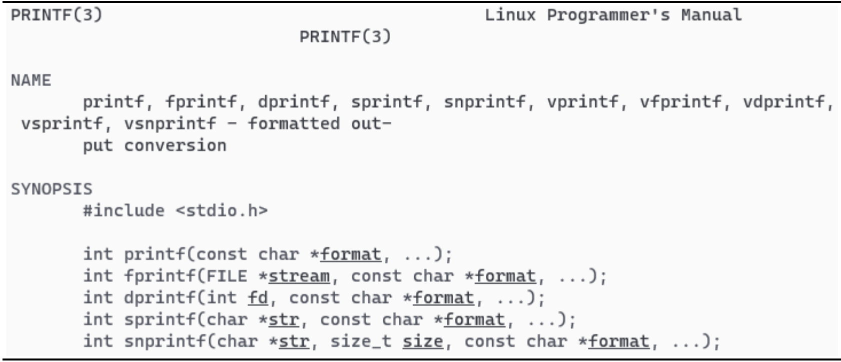
```
FGETC(3)                   Linux Programmer's Manual                  FGETC(3)

NAME
       fgetc, fgets, getc, getchar, ungetc - input of characters and strings

SYNOPSIS
       #include <stdio.h>

       int fgetc(FILE *stream);

       char *fgets(char *s, int size, FILE *stream);

       int getc(FILE *stream);

       int getchar(void);

       int ungetc(int c, FILE *stream);
```



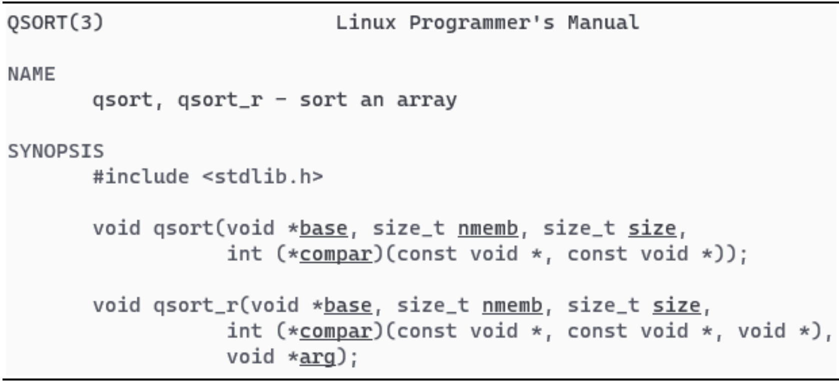
```
FORK(2)                                    Linux Programmer's Manual

NAME
       fork - create a child process

SYNOPSIS
       #include <sys/types.h>
       #include <unistd.h>

       pid_t fork(void);
```



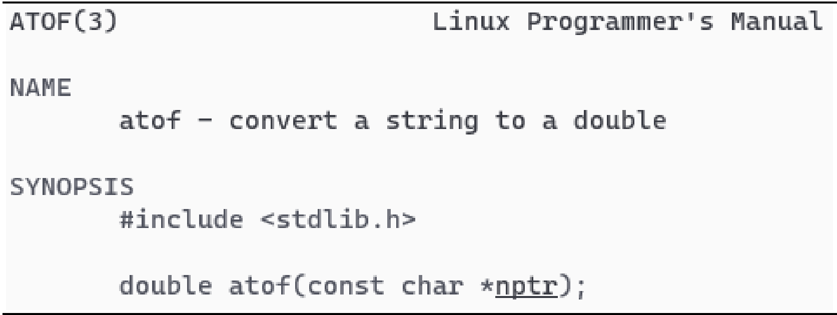
```
WAIT(2)                                    Linux Programmer's Manual

NAME
       wait, waitpid, waitid - wait for process to change state

SYNOPSIS
       #include <sys/types.h>
       #include <sys/wait.h>

       pid_t wait(int *wstatus);

       pid_t waitpid(pid_t pid, int *wstatus, int options);

       int waitid(idtype_t idtype, id_t id, siginfo_t *infop, int options);
                       /* This is the glibc and POSIX interface; see
                          NOTES for information on the raw system call. */
```



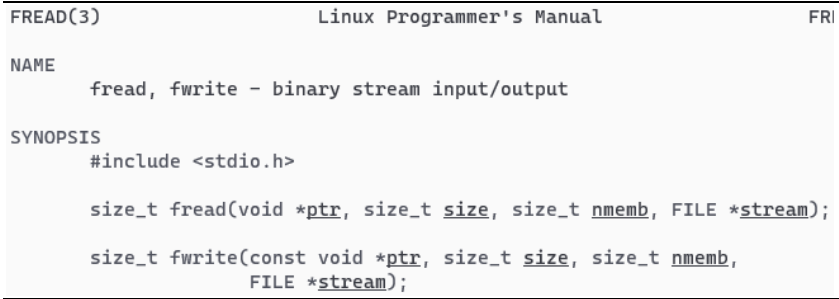
```
PRINTF(3)                                   Linux Programmer's Manual
                           PRINTF(3)

NAME
       printf, fprintf, dprintf, sprintf, snprintf, vprintf, vfprintf, vdprintf,
 vsprintf, vsnprintf - formatted out-
       put conversion

SYNOPSIS
       #include <stdio.h>

       int printf(const char *format, ...);
       int fprintf(FILE *stream, const char *format, ...);
       int dprintf(int fd, const char *format, ...);
       int sprintf(char *str, const char *format, ...);
       int snprintf(char *str, size_t size, const char *format, ...);
```



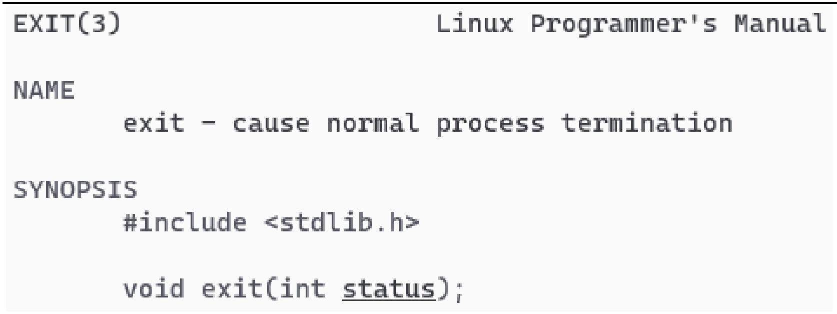
```
QSORT(3)                    Linux Programmer's Manual

NAME
       qsort, qsort_r - sort an array

SYNOPSIS
       #include <stdlib.h>

       void qsort(void *base, size_t nmemb, size_t size,
                  int (*compar)(const void *, const void *));

       void qsort_r(void *base, size_t nmemb, size_t size,
                  int (*compar)(const void *, const void *, void *),
                  void *arg);
```



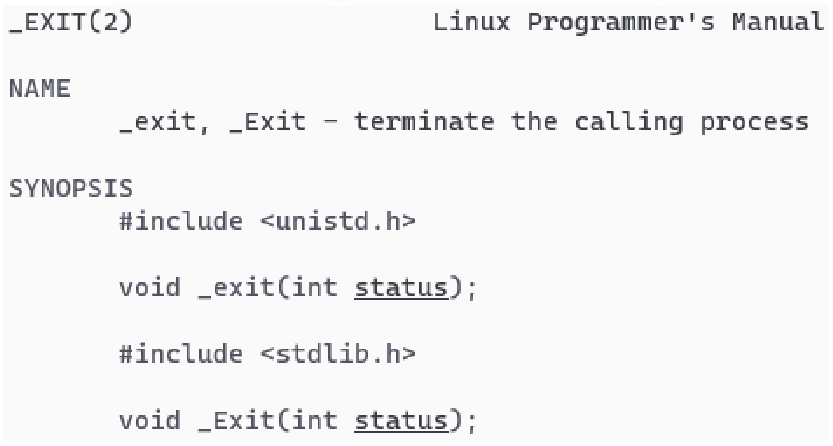
```
ATOF(3)                     Linux Programmer's Manual

NAME
       atof - convert a string to a double

SYNOPSIS
       #include <stdlib.h>

       double atof(const char *nptr);
```



```
FREAD(3)                    Linux Programmer's Manual                  FR

NAME
       fread, fwrite - binary stream input/output

SYNOPSIS
       #include <stdio.h>

       size_t fread(void *ptr, size_t size, size_t nmemb, FILE *stream);

       size_t fwrite(const void *ptr, size_t size, size_t nmemb,
                     FILE *stream);
```



```
EXIT(3)                     Linux Programmer's Manual

NAME
       exit - cause normal process termination

SYNOPSIS
       #include <stdlib.h>

       void exit(int status);
```



```
_EXIT(2)                   Linux Programmer's Manual

NAME
       _exit, _Exit - terminate the calling process

SYNOPSIS
       #include <unistd.h>

       void _exit(int status);

       #include <stdlib.h>

       void _Exit(int status);
```

SOLUTIONS

TEST

1	2	3	4	5	6	7	8	9	10	11	12
C*	B	B	C	A	A	A	C	C	A	A	C

*Option D is also accepted in the exam as the Aula Global version has #include <ststdio.h> instead of #include <stdio.h> because of an error when copying the code and, with this error, the code does not compile.

PROBLEM 1 - SECTION 1.1 AND 1.2

```
/*---------------------Libraries------------------------------------*/
#include <stdio.h>
#include <stdlib.h>
#include <errno.h>
#include <string.h>

/*---------------------Struct----------------------------------------*/
struct Student{
        char *name;
        int age;
};

/*---------------------Headers---------------------------------------*/
void print_struct(int i, struct Student *p_student);
void part_a(int p_age, int p_num);

/*---------------------Functions-------------------------------------*/
void print_struct(int student_id, struct Student *p_student){
        printf("Student id: %d\n", student_id);
        printf("Name: %sAge: %d\n", p_student->name, p_student->age);
}

void part_a(int p_age, int p_num){
         //For each student
         for(int i=0;i<p_num;i++){
                 char *line = NULL;
                 size_t len = 0;
                 ssize_t bytes_read;
                 printf("Write the name: ");
                 //Read the name
                 bytes_read = getline(&line, &len, stdin);

                 //Error in getline
                 if (bytes_read<0){

                 perror("ERROR IN THE INPUT"); exit(EXIT_FAILURE);}

                 else{
                         //Create element
                         struct Student *p_student;
                         //Design memory to the element
```

```c
                        p_student = malloc(sizeof(struct Student));
                        p_student->name=line;
                        p_student->age=p_age;
                        //-----Print function----------------
                        int student_id = i+1;
                        print_struct(student_id, p_student);
                        //Free the buffer in getline

                        free(line);
                }
        }
}

/*---------------------Main-----------------------------------------*/
int main(int argc, char **argv){
    int p_num = 0;
    int p_age = 0;

    //-------Check number of arguments--------------------------
    if ((argc < 2) || (argc > 3)){
         printf("Número de argumentos inadecuado\n");
         return 1;
    }

    else if (argc == 2){
         p_age = atoi(argv[1]);
         p_num = 3;
         printf("Two arguments. There are 3 students with %d years.\n",
p_age);
         }

    else{
         p_age = atoi(argv[1]);
         p_num = atoi(argv[2]);
         printf("Three arguments. There are %d students with %d
years.\n", p_num, p_age);

         //------Part A function---------------------------------
         part_a(p_age, p_num);
    }
         return(0);
}
```

PROBLEM 1 - SECTION 1.3

```c
/*---------------------Libraries-------------------------------------*/
#include <stdio.h>
#include <stdlib.h>
#include <errno.h>
```

```

/*----------------------Headers-------------------------------------*/
void print_array(int *array, int size);

/*----------------------Functions-----------------------------------*/
void print_array(int *array, int size){
  printf("{");
  for (int i=0; i< size-1; i++)
        printf("%i, ", array[i]);
  printf("%i}\n", array[size-1]);
}

/*----------------------Main-----------------------------------------*/
int main(int argc, char *argv[]){
    //------Assign memory array size 3--------
    int size = 3;
    int *first_values= NULL;
    first_values = (int *) malloc (sizeof(int)* size);

    //Fill the array
    first_values[0] = 20;
    for (int i = 1; i < size; i++)
         first_values[i] = first_values[i-1] + i;

    //Print array
    print_array(first_values, size);

    //------Reallocate memory array size 6------
    int new_size = 6;
    int *second_values = NULL;
    second_values= (int *) realloc (first_values,new_size*sizeof(int));
    first_values = second_values;

    //Fill end of array
    for (int i = size; i < new_size; i++)
         first_values[i] = i;

    //Print new array
    print_array(first_values, new_size);

    //------Free memory--------------------
    free(first_values);
         return(0);
}
```

PROBLEM 2

```c
#include <sys/wait.h>
#include <unistd.h>
#include <stdio.h>
#include <stdlib.h>

int get_number_of_bytes( const char *filename )
{
    int count = 0;

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

    if ( f == NULL ) return -1;

    int c = fgetc(f);
    while ( c >= 0 )
          {
          count++;
          c = fgetc(f);
          }

    fclose(f);

    return count;
}

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

    // this is only for part 3
    /*
    int process_count = 0;
    */

    for ( int i = 1; i < argc; i++ )
        {
        const char *filename = argv[i];

        pid = fork();

        if ( pid == 0 )
           {
           int count = get_number_of_bytes( filename );

           if ( count < 0 )
              {
              printf( "Error processing file %s\n", filename );
              _exit( EXIT_FAILURE );
              }

           printf( "File %s contains %d byte(s).\n", filename, count );
           _exit( EXIT_SUCCESS );
           }

        if ( pid < 0 )
           {
           printf( "Error creating process for file %s\n", filename );
           }

        // this is only for part 3
        /*
        else {
             process_count ++;
             if ( process_count == 3 )
                {
                wait(NULL);
                process_count--;
                }
             }
        */
        }

    pid = wait(NULL);
    while ( pid > 0 )
          {
          /* Process with pid 'pid' ended */
          pid = wait(NULL);
          }

    return EXIT_SUCCESS;
}
```

Assessment criteria

PROBLEM 1 (3.5 points)

- 1.1 Variable Parameters (1.75 points, 5 points out of 10)
  - Libraries
  - 0.35 p =Check number of arguments (2 out of 10)
  - 0.35 p =Declaration and initialization of the variables (2 out of 10)
  - 0.35 p =Conversion from parameters to integers (2 out of 10)
  - 0.35 p =Print error and the messages (2 out of 10)
  - 0.35 p =Call to part_a() (2 out of 10)
- 1.2 Structure and getline (0.875 points, 2.5 out of 10)
  - 0.0875 p =Struct definition (1 out of 10)
  - 0.175 p =print_struct definition (2 out of 10)
  - 0.175 p =Correct use of getline (2 out of 10)
  - 0.175 p =Creation of structure and malloc (2 out of 10)
  - 0.175 p =Assign value to the struct (2 out of 10)
  - Print struct
  - 0.0875 p =Free memory (1 out of 10)
- 1.3 Dynamic Integer Array (0.875 points, 2.5 out of 10)
  - Libraries
  - 0.175 p =Array creation and initialization (2 out of 10)
  - 0.175 p =Memory assignation size 3 (2 out of 10)
  - 0.175 p =Fill the array and print it (2 out of 10)
  - 0.175 p =Reallocate memory size 6 (2 out of 10)
  - 0.175 p =Free memory (2 out of 10)
- Significant errors may be subject to additional penalties.

PROBLEM 2 (3.5 points)

The grading of each part of the exercise is specified along its statement. Additionally, the following considerations will apply, during the appraisal, that might lower the qualification:

- Misuse of separators: semicolon, brackets, braces ...
- Misuse of parentheses when calling functions or brackets when using subindices.
- Misuse of operator "->" (arrow) or "." (dot) when accessing fields in compound data.
- Using uninitialized pointers, or wrong-valued pointers.
- Not freeing or misusing dynamic memory (doing this systematically is worse than incidentally, of course).
- Not freeing or misusing resource related to files (pointers to *FILE* or *handles*), or whatever other resources obtained from the operating system.
- Not initializing local (stack) variables.

Doing repeatedly any of these actions worsens the grading, but not proportionally.