Heterogeneous Data Structures and Dynamic Memory in C and Assembly

Luís Nogueira Paulo Ferreira Raquel Faria
André Andrade Carlos Gonçalves Marta Fernandes
Ênio Filho André Pedro

{lmn,pdf,arf,lao,cag,ald, evf,mpd}@isep.ipp.pt

2020/2021

Notes:

- Each exercise should be solved in a modular fashion, organized in a set of files and with an associated Makefile;
- The source code must be commented and have a consistent indentation;
- Ensure a correct manipulation of all heap blocks used in each exercise;
- Exercises should all be placed in the `modulo5` folder.

1. Consider the following data types:

```c
union union_u1{
   char vec[32];
   float a;
   int b;
} u;
```

```c
struct struct_s1{
   char vec[32];
   float a;
   int b;
} s;
```

   Create a function in C that prints the sizes of `u` and `s`. Explain the obtained result.

2. Consider the following program:

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

int main( void ){

   union union_u1{
      char vec[32];
      float a;
      int b;
   } u;
```

```
    union union_u1 * ptr = &u;

    strcpy(ptr->vec, "arquitectura de computadores" );
    printf( "[1]=%s\n", ptr->vec );
    ptr->a = 123.5;
    printf( "[2]=%f\n", ptr->a );
    ptr->b = 2;
    printf( "[3]=%d\n", ptr->b );

    printf( "[1]=%s\n", ptr->vec );
    printf( "[2]=%f\n", ptr->a );
    printf( "[3]=%d\n", ptr->b );
    return 0;
}
```

a) Explain the output.
b) Redefine `u` as a struct, run the program again and explain the new output.

3. Consider the following data type:

```
typedef struct {
  char age;
  short number;
  int grades[10];
  char name[80];
  char address[120];
} Student;
```

Develop in C a program that:

- Statically allocates an array with 5 elements of type `Student`;
- Implements a function `void fill_student(Student *s, char age, short number, char *name, char *address)` that saves the age, number, name, and address of a student in the fields of a structure whose address is given in `s`.
- Uses the function developed in the previous exercise to initialize the data for each of the structures in that array (consider that you don't have to initialize the grade's array for each student).

4. Extend the previous program and develop in Assembly the function `void update_address(Student *s, char *new_address)` that copies the string `new_address` to the field `address` of the structure pointed by `s`.

5. Extend the previous program and develop in Assembly the function `void update_grades(Student *s, int *new_grades)` that that copies all the elements of the array `new_grades` to the field `grades` of the structure pointed by `s`. Assume that the array `new_grades` has 10 elements.

6. Extend the previous program and develop in Assembly the function `int locate_greater(Student *s, int minimum, int *greater_grades)`. The function should determine which student's grades are greater than the value passed in `minimum`. All grades that satisfy this condition should be saved in the array pointed by `greater_grades`.
   - The function should return the number of grades added to `greater_grades`;
   - Assume that the array `greater_grades` is a statically defined array with 10 integers.

7. Consider the following data type:

```
typedef struct {
  int i;
  char c;
  int j;
  char d;
} s1;
```

Develop in Assembly the function `void fill_s1(s1 *s, int vi, char vc, int vj, char vd)` that saves the values passed in the function's parameters in the respective fields of the structure pointed by `s`. Take into account the memory alignment details in IA32/Linux.

8. Consider the following data type:

```
typedef struct {
  short w[3];
  int j;
  char c[3];
} s2;
```

Develop in Assembly the function `void fill_s2(s2 *s, short vw[3], int vj, char vc[3])` that saves the values passed in the function's parameters in the respective fields of the structure pointed by `s`. Take into account the memory alignment details in IA32/Linux.

9. Consider the following data types:

```
typedef struct {
   short x;
   int y;




} structA;
```

```
typedef struct {
   structA a;
   structA *b;
   int x;
   char c;
   int y;
   char e[3];
   short z;
} structB;
```

Implement the following functions in Assembly:

```
short fun1(structB *s){
   return s->a.x;
}

short fun2(structB *s){
   return s->z;
}
```

```
short* fun3(structB *s){
   return &s->z;
}

short fun4(structB *s){
   return s->b->x;
}
```

10. Develop in C the function `char *new_str(char str[80])` that dynamically creates a new string with the same content of the one that is received as a parameter but whose size is only the necessary to store that content. The function should return the address of the new string. **Explain why it is possible to return such address**.

11. Develop in C the function `int **new_matrix(int lines, int columns)` that dynamically reserves a memory block to hold a matrix `lines x columns`. The function should return the address of the newly created matrix.

12. Develop in C the function `int find_matrix(int **m, int y, int k, int num)` that searches the number received in `num` in a `y x k` matrix pointed by `m`. The function should return 1 if such value exists in `m` or 0, otherwise. Use the function developed in the previous exercise to create the matrix and fill it with random values.

13. Develop in Assembly the function `int count_odd_matrix(int **m, int y, int k)` that counts the number of odd elements in a `YxK` matrix pointed by `m`. Avoid the use of a division operation to determine if an element of `m` is odd or even.

14. Develop in C the function `int **add_matrixes(int **a, int **b, int y, int k)` that sums two matrixes `y x k` and stores the result in a dynamically created matrix whose address should be returned.

15. Implement in C the needed functions and data structures to simulate a stack using dynamic memory. The stack's size should reflect each `push` and `pop` operations. You can assume that only integer values are stored in this stack.

16. Consider the following data type:

```
typedef struct{
   int n_students;
   unsigned char *individual_grades;
}group;
```

Consider that a group has a number of students given by `n_students` and that `unsigned char *individual_grades` points to a dynamically allocated array of that size with the obtained grades of each student during the semester. Consider that, for each student, the 8 bits of the grade indicate whether or not the student was approved (bit at 1 or 0, respectively), in each of the 8 modules of continuous assessment during the semester.

Develop in Assembly the function `int approved_semester(group *g)` that calculates the number of students of a group that were approved in the semester. Consider that, in order to be approved, a student had to be approved in, at least, 5 modules.

17. Consider the following data types:

```
typedef union {
  int a;
  char b;
  short c;
  long int d;


}unionB;
```

```
typedef struct {
   short int a[3];
   char b;
   long long int c;
   int d;
   unionB ub;
   char e;
}structA;
```

Implement the following function in Assembly. Assume that *matrix* is a matrix of structures of type `structA` dynamically reserved in the heap with 4 lines and 3 columns.

```
char return_unionB_b(structA **matrix, int i, int j){
   return matrix[i][j].ub.b;
}
```