1 Pointers and variables

Exercise 1:

1. What are the values of !a, !b, !c after the following Caml code is executed?
   
   ```caml
   let a = ref 2;;
   let b = ref (!a);;
   let c = a;;
   a:= 9;;
   ```

2. Same question with a^, b^, c^ and the following Pascal code:
   
   ```pascal
   var a, b, c : integer^;
   new a; a^ := 2;
   new b; b^ := a^;
   c := a;
   a^ := 9;
   ```

3. Same with *a, *b, *c and the following C code:
   
   ```c
   int *a, *b, *c;
   a = malloc (sizeof (int));  *a = 2;
   b = malloc (sizeof (int));  *b = *a;
   c = a;
   *a = 9;
   ```

4. Same with a.get(), b.get(), c.get() in Python:
   
   ```python
   class ref:
       def __init__(self, obj): self.obj = obj
       def get(self): return self.obj
       def copy(self): return ref(self.obj)
       def set(self, obj): self.obj = obj
   a = ref 2
   b = a.copy()
   c = a
   a.set(9)
   ```

Exercise 2: Structures in C

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

struct personT {
    char name[32];
    int age;
```
// a function prototype:
void crazy_function(struct personT per, int a[]);

int main() {

    struct personT person;
    int i;
    int arr[5];

    for(i=0; i < 5; i++) {
        arr[i] = i;
    }

    strcpy(person.name, "Tia");
    person.age = 21;

    crazy_function(person, arr);

    for(i=0; i < 5; i++) {
        printf("arr[%d] = %d\n", i, arr[i], 5);
    }

    printf("age = %d name = %s\n", person.age, person.name);
}

void crazy_function(struct personT per, int a[]) {
    int i;

    for(i=0; i < 5; i++) {
        a[i] = a[i]*a[i];
    }
    strcpy(per.name, "Ace");
    per.age = 18;

    //DRAW THE STACK BEFORE THE RETURN STATEMENT IS CALLED
    return;
}

1. What is the output of the program?
2. Draw the stack contents just before the return statement of crazy_function is called.

Exercise 3: Pointers on structures
We define the following structures in C.

    struct s1 { int i; };
    struct s2 { struct s1 s; };
    struct s3 { struct s1 *p; };

1. Write two fragments of C code which create structures of the type struct s2, resp. struct s3, containing a structure with the value 42.
2. Draw a box-arrows diagram of memory after executing the following code:

    struct s1 s1; // yes, we can give the same name to
    struct s2 s2; // a variable and a struct type...
    struct s3 s3, ss3;
s1.i = 42; s2.s = s1;
s3.p = &s1;
ss3.p = malloc(sizeof(struct s1)); ss3.p->i = 54;

We will differentiate the heap from the stack.

3. We define the following function:
   struct s1 *f(void) { // (void) = does not take an argument
      struct s1 s;
      s.i = 42;
      return &s;
   }

   What happens when you run this function? It helps to first answer the question: What
   is the point of malloc()?

4. What is the difference between both of the following structure declarations? The ques-
   tion asks about the way the objects of both types are represented in memory.
   struct info1 { int value; struct s1 s1; };
   struct info2 { int value; struct s1 *p1; };

5. One of the following two statements will be rejected by the C compiler:
   struct tree1 { int value; struct tree1 left, right; };
   struct tree2 { int value; struct tree2 *left, *right; };

   Which one? Why?

6. Why are two types of «inclusions» of structures (look at question 4) authorised in C?
   (Only the first one is actually referred to as an inclusion.)

Exercise 4: Memory blocks and strings

1. Write the function:
   
   void *memchr(const void *s, int c, size_t n);

   Here is an extract from the man page:
   DESCRIPTION
   The memchr() function locates the first occurrence of c (converted to an unsigned char) in string s.
   
   RETURN VALUES
   The memchr() function returns a pointer to the byte located, or NULL if no such byte exists within n bytes.

   The type size_t is an integer type, reserved for specifications of lengths. The type void * is a type of pointer to any type. The modifier const promises the compiler that the function will not modify anything pointed to by s.

2. Write the function:
   
   char *strchr (const char *s, int c);

   Here is an extract from the man page:
   DESCRIPTION
   The strchr() function locates the first occurrence of c (converted to a char) in the string pointed to by s.

   The terminating null character is considered to be part of the string; therefore if c is ‘\0’, the function locates the terminating ‘\0’.

   RETURN VALUES
   The function strchr() returns a pointer to the located character, or NULL if the character does not appear in the string.
3. Write the functions:

```c
char *strstr(const char *haystack, const char *needle);
void *memmem(const void *haystack, size_t haystacklen,
             const void *needle, size_t needlelen);
```

which return the first position of `needle` in `haystack`. (We are not looking for the most optimal code.)