UNIVERSITÉ DE FRIBOURG
UNIVERSITÄT FREIBURG

A. Schaller
16-896-375
Series 6

1 Pointers and Arrays + Stack and Heap Segments

Question 1. *Draw the stack and heap segments just when the PC register points to the last semicolon ; of the following compound statements (assuming that local arrays are stored in the heap segment):*

1.

```
int i;
int j = 1;
int *p = &j
int**q = &p
int ***r = &q
i = ***r + 1;
```

2.

```
int i;
int tab[3];
int *p = tab;
++p;
++p;
i = p - tab;
tab[0] = 1;
(tab+1)[0] = 2;
*p = 3;
```

3.

```
int *p = malloc(2*sizeof(int));
p[0]=4;
p[2]=5;
```

2 Complicated Declarations

Explain in your own words:

1. Declare and allocate a 2-dimensional array with 3 rows of 2 columns. In the end, will allocate $3 * 2 = 6$ spaces for *ints*.

```
int a[3][2];
```

2. Following the '*C programming Spiral Rule*'[1](alternatively to the hint[2]), b is an array of size 3 of pointers to *ints*. Meaning, when unreferencing the pointer, we will have access to an array holding a 3 integers.

[1] http://c-faq.com/decl/spiral.anderson.html (visited on March 2021)
[2] https://cdecl.org (visited on March 2021)

```c
int *b[3];
```

3. Still following the *Spiral Rule*[1], the variable c is a pointer to a function without any parameter returning an integer.

```c
int (*c)();
```

4. d holds a function which accept as parameter a pointer to a function (e) which doesn't accept any parameter and the pointed function e returns an integer, which finally d return an integer.

```c
int d(int (*e)());
```

5. f holds a function which doesn't accept any parameter and return a pointer to a function which also doesn't accept any parameter and return an int.

```c
int (*f())();
```

3 typedef Definitions

Question 1. *The following* `typedef` *define some very common new types. Indicate their names and their corresponding defined types.*

1. The following `typedef` named `stackt` is a synonym to pointer of type `void`.

```c
typedef void *stackt;
```

2. `fctInt_t` is a synonym for a pointer to a function which accept a parameter integer and return an integer.

```c
typedef int (*fctInt_t)(int);
```

3. `fct_gen` is a synonym for a pointer to a function which accept a pointer `void` as parameter and return a pointer `void`.

```c
typedef void *(*fct_gen)(void *);
```

4. `signal` is a synonym for a function which accept 2 parameters: 1. an integer, 2. a pointer to a function which accepted an integer and return `void`, and return a pointer to function accepting a integer and return `void`.

```c
typedef void ( *signal(int, void (*)(int)) )(int);
// typedef void (* f_1)(int);
```

```
    // typedef f_1 signal(int, void (*)(int));
```

4 Pointer to Function + typedef

Consider the following program

```
typedef int (*mathFunc_t)(int, int); // definition of type mathFunc_t

int add(int a, int b) {
    return a + b;
}

int mult(int a, int b) {
    return a * b;
}

int compute(mathFunc_t f, int a, int b) {
    return f(a, b);
}

int main() {
    mult(add(2, 4), 8);
    compute(mult, compute(add, 2, 4), 8);
    return 0;
}
```

1. **Question 1.** *What is the return value of mult() and compute()?*

   It prints twice the value **48**.

2. **Question 2.** *Draw the simplified stack segments when the PC register points respectively to the last semicolon ; of the function calls in line 1 and 2 of main().*