System-oriented Programming
Spring 2018

S04

Professor : Philippe Cudré-Mauroux
Assistant : Michael Luggen

Submitted by Sylvain Julmy

Note : the complete source file are available inside the zipped file.

Exercise 1

Figures 4 to **??** show the different diagram for exercise 1.

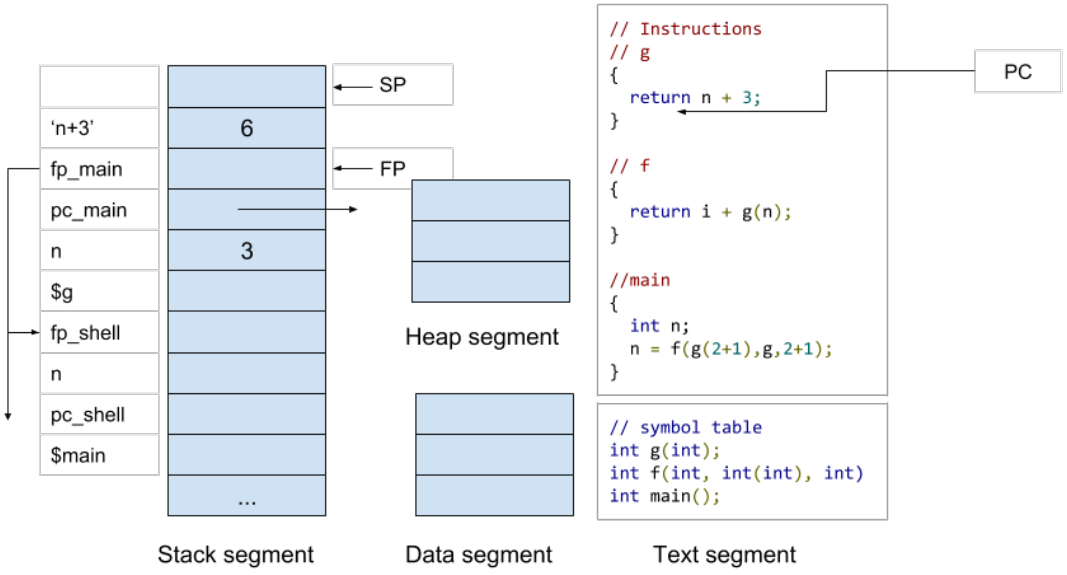

Figure 1: Exercise 1-a.

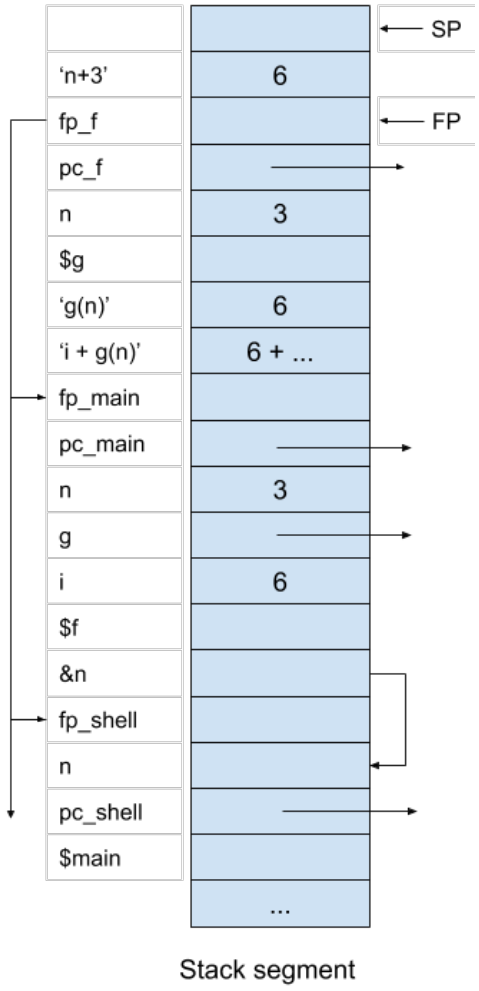

Figure 2: Exercise 1-b.

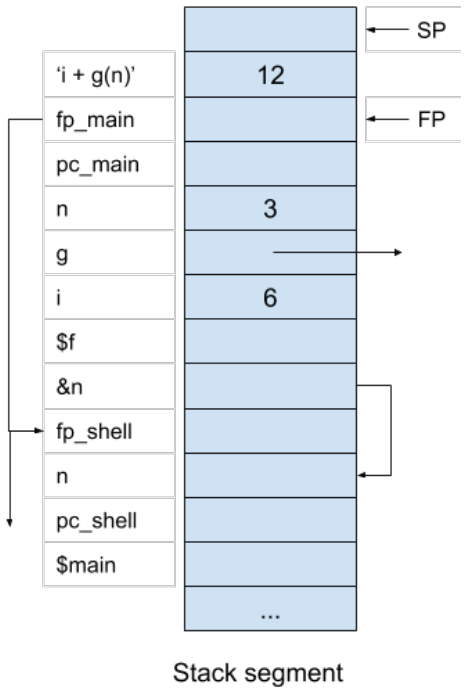

Figure 3: Exercise 1-c.

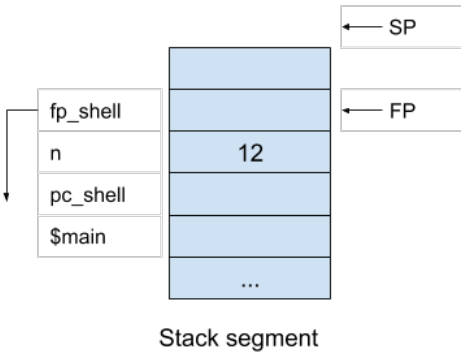

Figure 4: Exercise 1-d.

Exercise 2

a)

Figure 5 shows the simplified stack segment for point A.

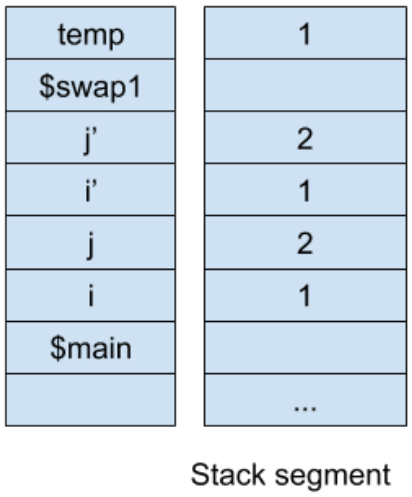

Figure 5: Exercise 2-a.

b)

Figure 6 shows the simplified stack segment for point B.

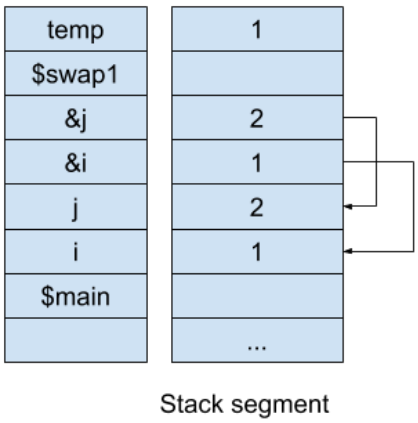

Figure 6: Exercise 2-b.

c)

Figure 7 shows the diagram for point A.
We use the following list of gdb command :

```
// breakpoint at function swap2
br swap2
```

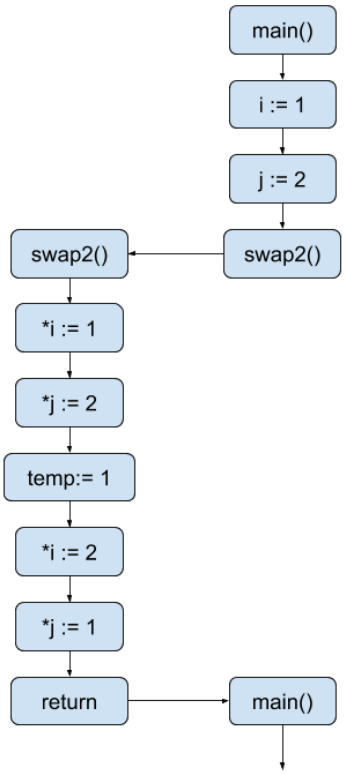

Figure 7: Exercise 2-c.

```
// breakpoint before returning from swap2
br 15
run

// display the variables
display i
display j
display temp

// display the arguments of stack2
frame 0

// continue to next breakpoint
c
// the value of i and j has been exchanged inside the function
n
// display the variables
display i
display j
// value have been exchanged
```

Exercise 3

Figure 8 show the stack and heap segment for a dynamic memory allocation.

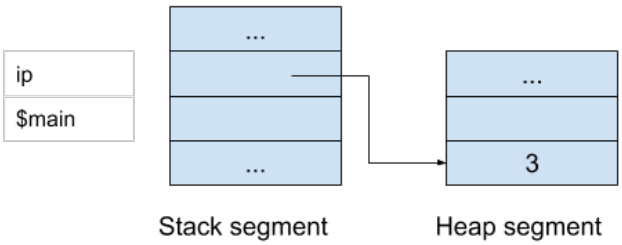

Figure 8: Stack and heap segment for a dynamic memory allocation.

The casting is not necessary because a `void*` is implicitely cast by the compiler.

Exercise 4

a)

The ternary operator evaluates only one of the 2 expressions depending on the value of the logical expression in front of the ?. So, if $bufp > 0$, $getChar()$ is never call.
This buffer is like a cache memory because, when we *ungetch* a character, we don't discard it but push it in the buffer. So multiple alternate call to *getch* and *ungetch* doesn't do any system call, which is clearly faster.

b)

Here is the new implementation for *getop* :

```
int getop(char s[])
{
    static int buf[BUFSIZE];
    static int bufp = 0;

    char c;
    int i = 0;

    while ((c = (bufp > 0 ? buf[--bufp] : getchar())) == ' ' || c == '\t'); // skip spaces

    if (c < '0' || c > '9')                    // c is not a digit
        return c;

    s[0] = c;
    while (isdigit(s[++i] = c = (bufp > 0 ? buf[bufp--] : getchar())))    // collect integer
        ;
    s[i] = '\0';                               // string terminator
    // save the last read character
    if (c != EOF)
        if (bufp < BUFSIZE)
            buf[bufp++] = c;
        else
            printf("%d %d error\n", bufp, BUFSIZE);
    return NUMBER;
}
```

c)

The variable are declared as static because they are "private" to the file. We don't want the other files to know about our variables or allow them to interact with our variables.

d)

The header file are here in order to "present" the API of our modules. The user of the file "stack.h", for example, don't have to know the exact details of the implementation, just what the module offer is enough.
The files don't have to be protected by `#ifndef` because, when the compiler is pre-processing the file, the `include` directives simply copy paste the whole header file inside an another one. Multiple declaration of the variable `stack_error` is not a problem as long as the type is the same. For example, declaration as follow generate a compilation error :

```
long stack_error;

int main(...)...
```

and a code like the following is accepted by the compiler :

```
#include <stdio.h>
#include <stdlib.h>^^I// atof

#include "./stack/stack.h"  // push, pop
#include "./stack/stack.h"  // push, pop
#include "./getop/getop.h"  // GETOP_ERR, MAXOP, NUMBER, getop

int stack_error;

// reverse Polish calculator --- kr76
main()...
```