Exercise 4

1. Reading (optional)

Browse/Read the chapter 4: from B. Kernighan, D. Ritchie, The C Programming Language, 2nd Ed., Prentice Hall, 1988.

2. Memory

Draw the memory content of the following program at the end of line 4 (assuming a 32-bit architecture):

```
int main() {
    int i=8;
    char c1='@';
    char c2='A';
    char s[5]="@A";
}
```

1. with symbolic names and values of the variables, (cf. Fig. 7)
2. same as (1), but augmented with symbolic names and possible hexadecimal values of the addresses, (cf. Fig. 6)
3. big and little endian, with symbolic memory cell and address values, (cf. Fig. 5)
4. big and little endian, with hexadecimal memory cell and address values, (cf. Fig. 4).

3. Control Flow & Abstract Syntax Tree

Consider the following program, draw the control flow for the function calls main() and swap1(i,j) and the associated AST, including the call stack just before swap1 returns to its caller function main – ignore the function call `swap2(&i, &j)`.

```
void swap1(int i, int j) {
    int temp;
    temp = i;
    i = j;
    j = temp;
}
void swap2(int *i, int *j){
    int temp;
    temp = *i;
    *i = *j;
    *j = temp;
}
```

```
int main(){
    int i;
    int j;
    i=1; j=2;
    swap1(i,j);
    swap2(&i,&j);
}
```

Check specifically if the values of i, j were swapped in main() and report your result.

4. Macros and C Preprocessor

1. Test the following scenario on machine and explain the output. **Hint:** Explain what is wrong with the macro 'square' and write a correct version.

```
#define max(A, B) ((A) > (B) ? (A) : (B))
#define square(x) x * x
max(a, b);
max(a+1, b+1);
square(x);
square(x+1);
```

2. Write a macro swap(t,x,y) that exchanges the values of the two variables x and y assuming that both are of type t, e.g. int, and test it on machine. **Hint:** Use a block structure and test your macro with the program:

```
#include <stdio.h>
#define swap(t,x,y) /* complete this macro */
int main() {
    int a=1;
    int b=2;
    swap(int,a,b);
    printf("%d %d\n", a, b);
}
```

3. (tricky) If you found a working solution for your macro swap(t,x,y) in the previous exercise, this solution will probably not work in the following situation:

```
if (a>b) swap(int,a,b); /* whoops */
else a = b;
```

Why? Hint: write down the code of line (1) once the macro has been expanded, and you will see the problem (if not, compile your code and understand the compiler's complaint).

Adapt the code of your macro in order that the above lines (1)-(2) become a correct C statement. Hint: the solution is tricky ! Nevertheless, try to find a solution by your own, e.g. without "google search"..

4. (tricky) A common approach to generating a single source code that is suitable for both development and release is done with the help of the following macro:

```
#ifdef DEBUG
# define DEBUG_PRINT(x) printf x
#else
# define DEBUG_PRINT(x) do {} while (0)
#endif
```

Use it like:

```
DEBUG_PRINT(("var1: %d; var2: %d; str: %s\n", var1, var2, str));
```

Write a little program to test it.

Note: the program must be compiled with the -D option to define DEBUG:

```
$ gcc -D DEBUG prog.c -o prog
```

5. Make

In this exercise we consider that the three files hello.c, tellMe.c and Makefile contain the following content.

hello.c:

```
void tellMe(void);
int main () {tellMe();}
```

tellMe.c:

```
#include
void tellMe () {printf("hello\n");}
```

Makefile:

```
hello : hello.o tellMe.o
gcc -o hello hello.o tellMe.o
hello.o : hello.c
gcc -c hello.c
tellMe.o : tellMe.c
gcc -c tellMe.c
```

and we consider the following scenario:

```
$ make # (1)
$ touch tellMe.c # or modify tellMe.c # (2)
$ make tellMe.o # (3)
$ make # (4)
$ make # sic!, redo a make # (5)
$ make -W tellMe.c # (6)
```

2. Draw the dependency tree of hello
3. What is the information displayed by the commands (1), (3)-(6)? Explain and verify your answer on machine

6. Git & GitLab

Create a new directory with the wcount.c example from Exercise 2.

Write a Makefile with the following functions:

- Compiles the wcount.c source file to a wcount binary
- Runs wcount with counting the wcount.c source file.

After everything compiles and runs accordingly:

Initialize a GIT repository in the directory with the above files:

```
$ git init
```

Add the Makefile and the wcount.c source. (Do not include the binary!)

```
$ git add Makefile wcount.c
$ git commit -m "Initial commit"
```

Create on the DIUF GitLab (only available if connected through the VPN) with your University credentials a Project called *SOP_Exercise4*.

Connect the local GIT repository to the new Project:

```
$ git remote add origin https://diuf-gitlab.unifr.ch/YOUR_USER/SOP_Exercise4.git
```

Upload your files to GitLab

```
$ git push -u origin master
```

Finally share your GitLab Project with the assistants of the course with Reporter access level:

→ Settings → Members → Invite Members.

Hand in.

Upload your answers on Moodle.