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

When we try to add an external variable of type `char` *c*, we got a compilation error when trying to linking (with `ld`) the object code, because the variable *c* don't exists.
We simply modify the source of `global2.c` by adding the following line in the code :

```
char c;
```

Then the compilation is succesfull because the linker have found the variable *c* in the object code.

Exercise 2

a)

Figures 1 shows the flow of the function call.
Figures 2 shows the AST of the function call, note that the declaration of the variable are normally not part of the AST because they are erase by the compiler because they are not allocated at runtime. They are here for information about the program.

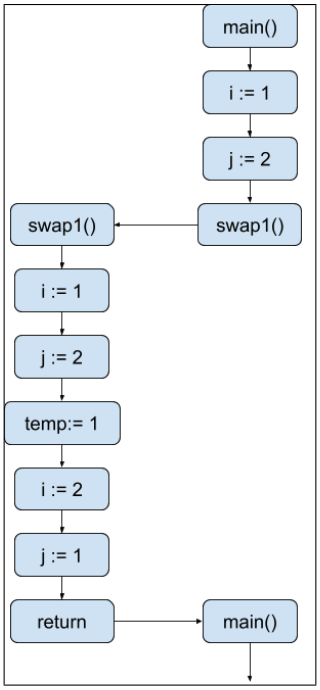


Figure 1: Flow of the function calls from Example 1 of GDB tutorial.

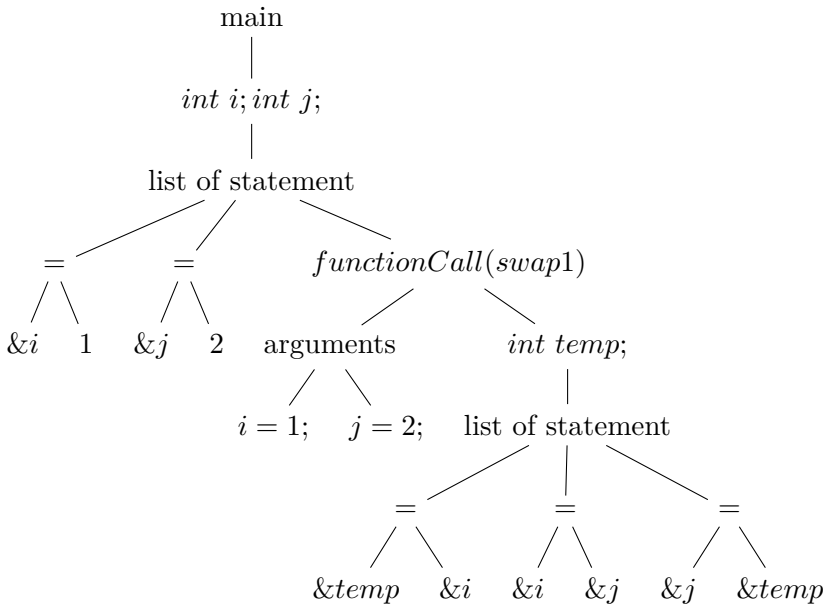


Figure 2: AST of the function main.

b)

We use the following list of gdb command :

```
// breakpoint at function swap1
br swap1
// breakpoint before returning from swap1
br 7
run

// display the variables
display i
display j
display temp

// display the arguments of stack1
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 hasn't been exchanged
```

The value of the variables are not exchanged, because the function *swap*1 is a call-by-value function and the value of i and j are copied on the stack. We have to use pointer in order to have a call-by-reference argument.

Exercise 3

a)

The macro definition is wrong because of the precedence of the operators in C. For example, if the use `square(4 + 1)`, this would expand to `4 + 1 * 4 + 1`, which would lead to an unexpected result. So we have to write the macro like the following :

```
#define square(x) ((x) * (x))
```

b)

```
// PRE : x and y are variable identifier of type Type
#define swap(x, y, Type) do{Type t = x; x=y;y=t;}while(0);
```

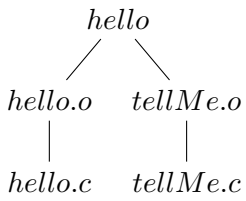
c)

Well, I do know the trick about using a `do while` construction in order to prevent bug...

The idea is to prevent unexpected statement precedence by using the mentionned construction. This does not affect the compiler because the code is simply erased by the compiler because the `do while` does nothing.

Exercise 4

a)



b)

Note : the warnings generated by the compiler has been removed.

(1)

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

(3)

```
gcc -c tellMe.c
```

(4)

```
gcc -o hello hello.o tellMe.o
```

(5)

```
// hello is already builded
```

(6)

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

c)

Figures 3 to 6 shows the execution trees of the corresponding command.

d)

`tellMe_unwiseBug.c`

There is no problem at the compilation or at running time.

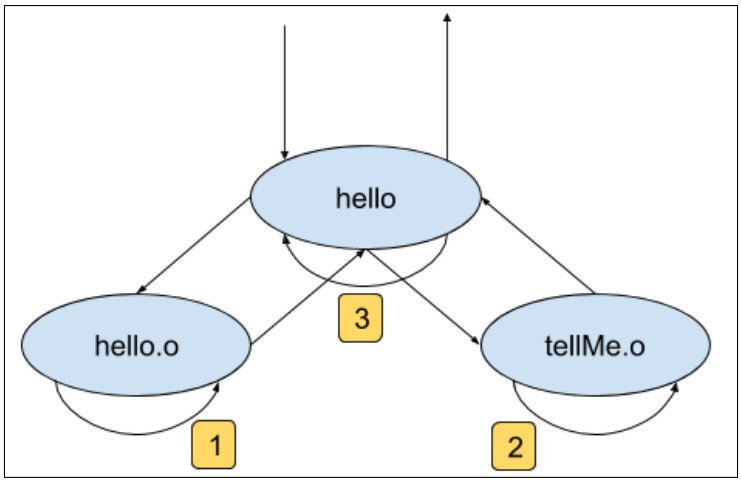


Figure 3: Execution tree of (1).

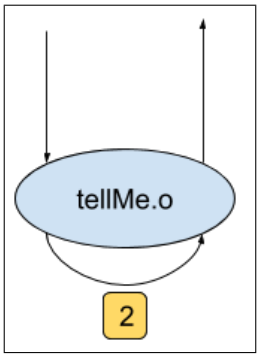


Figure 4: Execution tree of (3).

`tellMe_compBug.c`

There is an error at the compilation, because the function *tellMe* is declared with two differents signatures. Once, it has an argument of type *char* and another time it is declared with argument of type *char*[] (pointer of char).

`tellMe_execBug.c`

There is no problem at the compilation. At the running time, the problem is that *tellMe* is called with an incorrect argument (wrong type). The problem appears because we don't check function prototypes across compiled source files.

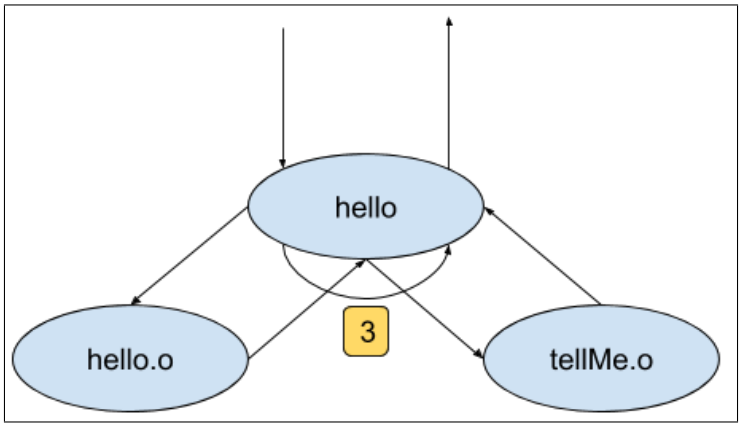


Figure 5: Execution tree of (4).

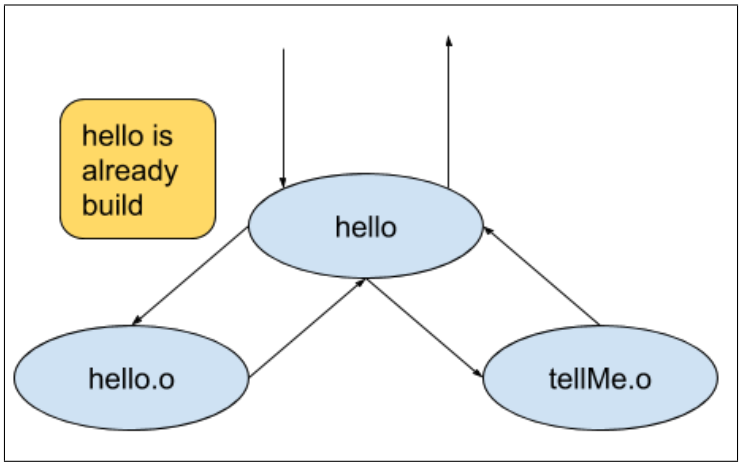


Figure 6: Execution tree of (5).