Tasks

Phil Reinartz

00.00.0000

1 Task 1 Stages of the compilation)

First stage "Preprocessing" The main function of the preprocessor is, that it adds the "full" c code into your .i data. For example, the stdio.h librarie is nothing other than some fuctions that someone declared, so that you don't have to write them yourself. These functions are loaded by the preprocessor in your programm, because you wrote #include <stdio.h> inside your code. The same is also for makros that you have placed. For example: if you wrote "#define Maximalwert 12345678909876543212345678999999999", the preprocessor takes the value of Maximalwert and puts it everywhere in the programm where you wrote Maximalwert. So all in all, the preprocessor does nothing other, than complete your programm by replacing predefined makros with the right value and adding and suptracting content you want to include, or your machine needs to work right. So that your programm is ready to be compiled in the next step.

Second stage, the compilation to assembly The next part of the compilation is the part where the most work gets done. In the first step, your preprocessor did necessary replacements for you, to make your .c script ready to be compiled. Now the compiler has to create a file, that your machine is able to process. The first step of this process is, to create assembly instructions of the preprocessors revised c code. The assembly instructions are a "humanoid" version of the binary, that is in some context also textbased, but on a very low leveltextbased. The now created .s file is fuction wise the equivalent of your .i file written in assembly. For example: the assembly instruction "movl $0, %eax" is the equivalent of setting the return value of your function by zero.

Third stage, creating the objectfile The object file is an interessting part of the compilation that is a easy to understand process, but not an easy to understand file. The object file is basically the functional equivalent of your assembly instructions, but written in machine language(not the full binary, but also writen with 0s and 1s). The .o file is hard to read, because you have to translate every line of code with the manual of your processor, because the binary language is architecture dependend. If you work with several different processors, this could get very complicated very quickly.

Final stage, linking your .o file. The final step of your compilation is, that your script needs somehow to be executable. This is done at the linking Stage, which works kind of similar to the preprocessing staged (don't get me wrong, it's definetelly not the same or similar thing, but the basic idea is similar). The linker is necessary, to define everything, that is still undefined, which could be for example some kind of machine specific function that needs to be declared. Otherwise, your machine probably wouldn't know what to do at the part where the function is used, because the machine has not the abillity to process other things than basic commands. So the linker replaces "more complex" functions with the address to the basic commands that describe the used function. This turns your file into an executable, that has several connections to many other .o files, that do nothing other, than describing functions of your .o file into processor understandable commands.

2 Task 2

1) I don't know if it's necessary to explain the cp (copie tool), but it's definetly an easy to use powerful tool to copie data.

The grep programm Grep is a powerfultool, which was made for searching textbased data. It's usefull for searching specific words, strings or patterns in a file. In the example we did: `grep -En printf\s*\(" solutions/debug_sample.c` grep is the programm, that we are using, wich is made for finding specific finding regex-based data. -E stands for "extended regular expressions", that means that differences for basic regular expressions are summarized afterwards. The -n stands for "line number", wich does prefix each line of output with it's line number from the file. `"printf\s*("` is the regex pattern, for that we are searching. `\s` is nothing other than a empty char " ", the * means that the preceeding item will be matched 0 or more times, wich defines, that grep would also find the printf if there is a tab behind. Last but not least is the `\(` which does declare the "(" as a character and not as an open expression. The Last part of the command is the place where we are doing the grep:`solutions/debug_sample.c`, wich in our case is in the `debug_sample.c` file. Honestly I forgot about grep, but I think it's yet another powerful tool, that we could use from the consol to find certain text patterns, which I gues would help us in the future with the next tasks.

The sed programm Sed (stream editor for filtering snd transforming text) is a stream editor, wich often is used to perform basic text transformations on an input stream. If you know what you are doing, than sed is way more efficient and faster than a normal editor, by manipulating the file from your Termal. In the example we did: `sed -E -i.bak 's/printf\s*/debug_printf/g' solutions/debug_sample.c`. The first part of the command is the programm that we are using, wich in our case is sed. Next part is the `-E`, wich as in grep stands for the

3 1.3)

Answer: Hpjhnefoisjhblsjhnbvosjhbvls,jhb