Assignment 1: Introduction to C++

January 27, 2016

1 git

In this section, you are familiarized with git repositories. Please use exactly the same file and directory names as mentioned below (including lowercase and uppercase letters). A script will automatically check your submission and any deviation from the given files, will lead to reduction of marks.

1. Refer to table 1 to look up your user name which you need to log into the server at `nanu.iiitd.edu.in`. Please note:
   - The server is only accessible through the IIITD Intranet and VPN.
   - Everybody has to do the assignment individually.
2. Log into the server using secure shell (SSH) using Putty (Windows) or `ssh` (Linux). You have to change the password the first time you do it. The standard password is the same as your group name.
3. After changing your password, log in again.
4. Create a directory `testRepo.git` and inside this directory an empty server sided git repository. If you have a look into `testRepo.git`, the following directory structure must have been created:

```
drwxr-xr-x  2 alefel alefel 4096 Mar  9  2015 branches
-rw-r--r--  1 alefel alefel   66 Mar  9  2015 config
-rw-r--r--  1 alefel alefel   73 Mar  9  2015 description
-rw-r--r--  1 alefel alefel   23 Mar  9  2015 HEAD
drwxr-xr-x  2 alefel alefel 4096 Mar  9  2015 hooks
drwxr-xr-x  2 alefel alefel 4096 Mar  9  2015 info
drwxr-xr-x 87 alefel alefel 4096 Nov  5 20:49 objects
drwxr-xr-x  4 alefel alefel 4096 Mar  9  2015 refs
```

Listing 1: Example of a server side git repository

5. While on the server, `clone` the repository to the directory `testRepoCloned`.
6. Clone the repository on your PC as well.
7. In this repository create a C-program `hello.c` which prints out `Hello, world!` in your terminal.
8. Add the file, commit the changes with a useful comment and push the new revision to the repository on the server. Do not commit the binary (executable) file. Binary files are architecture specific and an output of your compiler. Repositories cannot keep track of incremental changes of binary files and therefore any change/recompilation will copy the whole file content into the new repository consuming space unnecessary. Remember, once committed, a repository never forgets and a repository could contain thousands of revisions.
9. On the server, change into the directory `testRepoCloned`, update the repository and observe, if the file `hello.c` appears.
10. Check the `log` of the git repository to check the timestamp, author and comment of the current revision.

Username	Name
group01	Ricktam Kundu
group02	RAHUL GUPTA
group03	AARTIKA SETHI
group04	AKSHAYA BHARDWAJ
group05	ANDREW GIGIE
group06	ANKUSH SINGH
group07	ANURADHA
group08	CHAYAN PATHAK
group09	DEEPAYAN BANERJEE
group10	DIVYA JAIN
group11	HIMANI JAWA
group12	KUNWAR TARUN
group13	PANKHURI
group14	PARITOSH ASWINKUMAR JIVANI
group15	POOJA CHOUDHARY
group16	RAJENDRA PRASAD NAYAK
group17	RIMJHIM KHANDELWAL
group18	SAKSHI GARG
group19	SHIPRA BATRA
group20	SHIVAM KALLA
group21	SHRESTHA BANSAL
group22	VIKAS KIMAR
group23	VIPRA SHUKLA
group24	VISHAV VIKASH
group25	VIVEK TYAGI
group26	YATHARTH
group27	Niharika Agarwal
group28	Naina Gupta
group29	Payal Garg

Table 1: User names

2 Makefile

Create a Makefile to achieve the following:

1. Have a look at the Makefile Biryani uploaded to Backpack.
2. In `hello.c` (refer to section 1) remove the `main` function. Create a function `void hello()` instead, which prints the string `Hello world!` in your terminal.
3. Create a header file `hello.h` containing the function prototype `void hello()` and an include guard*. Include `hello.h` in `hello.c`.
4. Create a C file `main.c` which contains the `main` function and which includes only `hello.h` and not `hello.c`. The `main` calls `hello()` to print the text on the screen.
5. Create a Makefile, which is capable of compiling your program and to achieve the desired functionality:
   - Use variables for the compiler executable, flags, source files, file dependencies, and output file names.
   - Include a target `clean` in your Makefile which allows the user to delete all temporary files created during the compilation process, conveniently.

*`https://en.wikipedia.org/wiki/Include_guard`

- Make sure that the Makefile recompiles only those files, which have changed. If a programmer changes something in `main.c`, `hello.c` must not be recompiled. To achieve this, compile each C file into a separated object file first. As discussed in class the Linker can then take the already compiled objects (in this case `hello.o` compiled from `hello.c` earlier) and link it together with `main.o` which was recreated due to the file changes, to a new executable. There are some special variables (e.g. `$<`, `@<`,...) reducing your efforts.

3 ALU in CPP

3.1 System Architectural Model (SAM)

Create a System Architectural Model (SAM) or also called software based model of the ALU discussed in class. Implement the desired functionality in an object named `ALU`. The header file (`ALU.hpp`) is given in listing 2.

```
1  #ifndef ALU_HPP
2  #define ALU_HPP
3
4  #include <iostream>
5
6  using namespace std;
7
8  typedef enum {ADD, SUB, MUL, DIV, CMP} Operation;
9
10 class ALU {
11     public:
12         int execute(int op0, int op1, Operation operation);
13 };
14
15 #endif
```

Listing 2: `ALU.hpp`

1. Create the expected functionality in `ALU.cpp`.
2. Create `main.cpp` instantiating the class `ALU` and verify its correctness.
3. Create a Makefile which allows you to compile all C++ files into objects (e.g. `main.o`, `ALU.o`) before linking them together to an executable. Also make sure that only those files which were modified, are recompiled.
4. After everything is working, submit your code to the repository `assignment1.git` in your user directory.

3.2 Registers

1. Create a branch of your repository `assignment1.git` named `registers`. Continue with the assignment using this branch.
2. Extend the program implemented in section 3.1 by adding another class called `Registers` which contains a storage (array) for 16 values, each of them 32bit wide. Each of the value needs to be addressable from outside the class. So a good structure would be an array.
3. Modify the program further to accomplish the following:
   (a) `main` is not allowed to send the operands to the ALU directly. Instead it has only access to the storage provided by `Registers`.

(b) After loading the operands into the registers, extend the instructions in such a way that in addition to the operation to be executed, the ALU also gets to know, which registers to load the operands from. To achieve this, the class `Registers` need to be accessible from the class `ALU` (Hint: Give a reference of `Registers` to the constructor of `ALU`.)

(c) Modify the ALU such that the result is always stored in the register of the first operand. Choose a suitable format to transmit the register location information to the ALU.

4. Test the functionality by adopting `main.cpp` and the Makefile. Commit your changes to the repository regularly in the branch `registers`.

3.3 Load a Program

In your repository, create another branch from `registers` called `loadProgram`. Continue to work in this branch from now onwards.

Modify the program of section 3.2 by adding another class `LoadProgram`, which loads a program from an ASCII file. In this file, the first column contains the command to be executed, the second and third columns hold the first and second operand respectively separated by a white space (refer to example 3).

```
LOAD R0 4
ADD R0 6
SUB R1 R0
```

Listing 3: Example Program

`LoadProgram` loads the program from a file, stores each command into a suitable structure and saves the structures into a `vector` (refer to `http://www.cplusplus.com/reference/vector/vector/`). Change your program of section 3.2 to the following work flow:

1. When started, `LoadProgram` reads in the program file. There are two different instructions now:

   (a) Instructions such as `ADD`, `SUB`, `MUL`, `DIV`, `CMP` etc. are executed by the ALU.

   (b) Instructions for the control flow such as `JMP`, `LOAD`, etc are executed in the main (which essentially becomes a finite state machine).

2. The main requests the first program line, and depending on the instruction, sends it to the registers, ALU, or performs an control flow operation. For instance the `JMP` (jump) instruction will alter the Program Counter (PC) which essentially tells the `LoadProgram` class, which line to return next.

To complete this assignment successfully, create an ASCII file for the program given in listing 4. Extend the instruction space for the ALU and FSM. Only a subset of the instruction set mentioned in the manual of the M32 simulator (`http://nanu.iiitd.edu.in/AELD/M32.zip`) must be supported by your program with two exceptions:

1. If `COUTALL` is encountered, the class `Registers` returns the values of all its registers (good for debugging purposes).

2. If `COUT R`n is encountered, the class `Registers` returns the value of the register Rn.

```
#include<stdio.h>

int main() {
    int n=10, first=0, second=1, next, c;

    for(c=0;c<n; c++) {
        if(c<=1) {
            next = c;
        } else {
            next = first + second;
            first = second;
```

```
12                second = next;
13            }
14        }
15        return 0;
16    }
```

Listing 4: Program

4 Documentation

After completion of the assignment, create a documentation which shows the relationships among the classes you have implemented, using UML. Export this document as PDF and upload it to Backpack before the deadline. Any other document type apart from PDF is not considered, since no proprietary software is available to open e.g. Microsoft® Word documents.

5 Remarks

- Do not forget to submit (push) your final additions to the repositories before the deadline.
- If a name is given for e.g. a class or repository, please use it in exactly the same way. Pay attention to uppercase and lowercase spellings.
- If you plagiarize, the policy of the institute will be applied (`https://www.iiitd.ac.in/education/resources/academic-dishonesty`) with the exception that apart that your assignment will not carry any marks, your final grade is going to be reduced by one grade point instead of a letter grade. You may discuss possible solutions, however refrain from inspecting and/or copying code from websites, colleagues, etc.
- Regardless of your system, the submitted code is checked out and compiled on `nanu.iiitd.edu.in`. Therefore you have to make sure, that your Makefiles are correct and that your program compiles on the server. So after a submission, you can log in to the server using SSH, check out a local copy of the repository (refer to section 1) and execute the Makefile.
- Only the last push towards the repository on the server before the deadline expired, is evaluated.