Task documentation CLS: C++ classes in Python 3 for IPP 2015/2016
Name and surname: Matej Marušák
Login: xmarus06

1 Task

The goal was to create a script for inheritance analysis of C++11 classes. The script outputs inheritance tree of selected classes or details of all members in specific class.

2 Solution

Firstly, a simple error handler was created. After calling this function, an adequate error message is printed to stderr and the whole script is exited with a suitable exit code. In this text, *ERROR* refers to this function being called.
The main script runs following 4 steps in the given order.

1. **Parse command line options**
   For this task a simple command line parser was written. Firstly, the number of parameters is counted. If the number is not in range $\langle 1, 5 \rangle$, ERROR is called. Next, each argument is compared with possible arguments, that means, is checked if starts with one from this list ("--conflicts", "--details","--input=","--output=", "--help", "search=", "pretty-xml"). If so, this fact is marked down and possible arguments were parsed and saved as well.
   This part also checks for duplicity arguments and prohibited combinations.

2. **Processing input file**
   Input file obtained from previous step is opened. If the file name was not specified, stdin is considered. Next, function for parsing classes is called. This function takes reference to an opened stream. Output that is returned is dictionary of all the classes. Structure of this associative array is shown in figure 1.

   Process of creating returned structure is similar to the parser from compilers. Function GetToken() was created for recognizing lexical elements. This function is based on one regular expression. In a while loop a top-down parsing is performed. When unexpected token appears in parsing ERROR is called. Also in the process of parsing input a numerous checks are performed.

3. **Complete classes**
   Process of completing classes means adding all members from base classes to the child class. Each and every element is tested whether adding it would not cause conflict, if not some items may be edited before adding (e.g privacy). If item could cause conflict, this member is added to the list *conflicts*.

   At the beginning of this action, all classes are sorted to two lists - to the list *Closed* are added all classes that do not have any base classes. Classes that inherit from another class are added to list *Open*.

   Next a class is selected from *Open*, which all base classes are in *Closed*. This class is processed and removed from *Open* and added to the *Closed*. This process is looped until *Open* is empty.

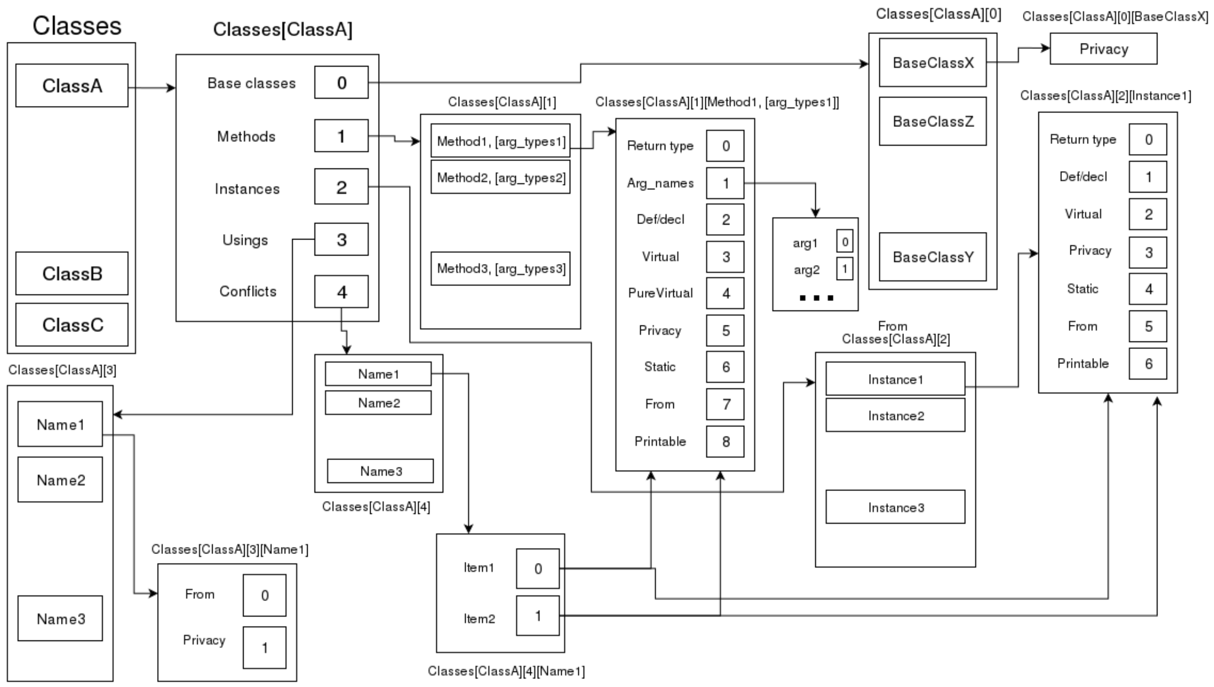

Figure 1: Strucuture of parsed classes

4. **Create output and write it** Firstly options are checked and whether option *–conflicts* is specified or not. If it is not and any conflict were found in the previous step, ERROR is called.

   Secondly the option *–details* is checked and based on it's presence different XML structures are created.

   Creating an XML is straightforward when either lxml or xml.etree library is used. Therefore this action will not be discussed further. Output file name obtained from step 1 is opened and the result from previous step is written into this file. If the file name was not specified a stdout is considered. When the file cannot be opened or written into, ERROR is called.

3 Extensions

A **CFL** extension was implemented. To make this extension, only a function for making different XML was created. This function takes dictionary of conflicts that was created in step 3 and can be seen on figure 1.

4 Conclusion

Created script was tested by official test. Another 50 tests were created. To test these tests a bash script for automatic runing test and checking output was made. All tests passed successfully. A script was tested on school server merlin as well.