CSE 756, Project 5: Intermediate Code for Statements

You need to extend your solution for Project 4 to translate Sage AST statements to intermediate code. As with Project 4, this will be done by traversing the Sage AST in ROSE, extracting the necessary information from the AST, and constructing a string that contains the resulting transformed program. This string will be printed by **`main`** (already provided in Project 4; do **not** change it). The result should be a valid C program that is equivalent to the input program. The project is due by **May 17 (Thursday)**, 11:59 pm.

Input Language

Assume that the input is a C program that satisfies all restrictions described in Project 4, with the following modifications:

- We allow again the relational operators (Section 6.5.8) **>**, **>=**, **<**, and **<=**. However, the only expressions that contain these operators are the ones used in conditions of loop statements and if statements.
- We also allow the remaining operators for boolean expressions: **==** and **!=** (equality operators; Section 6.5.9), **&&** and **||** (logical operators; Section 6.5.13 and Section 6.5.14), and **!** (logical negation; Section 6.5.3). As with the relational operators, the only expressions that contain these operators are the ones used in conditions of loop statements and if statements.
- The grammar discussed in class assumes boolean literals **true** and **false**. C does not have these; so, you can ignore these productions of the grammar.

Translation

You need to translate all statements including sequences (i.e., SgBasicBlock), if-statements and if-then-else statements, while-do loops, do-while loops, for loops, expression statements, and return statements. For expression statements and return statements, use your solution from Project 4. You have to generate intermediate code for each such statement, using the approach discussed in class. The intermediate language is the one described in the lecture notes. Since this intermediate language is a proper subset of C, your output program will be a C program. For example, if the input is

```
main(){
      int x = -1, y;
      if (x+1 < 0) y = 1;
}
```

the resulting program could be something like

```
main(){
      int _t1;
      int x = -1, y;
      _t1 = x + 1;
      if (_t1 < 0) goto _l1;
      goto _l2;
      _l1: y = 1;
      _l2: ;
}
```

This is valid C code. In the input program, the boolean conditions in ifs and loops could be arbitrarily complicated, including relational/equality operators with operands that themselves are complex sub-expressions with **+**, *****, etc. Temporary variables need to be introduced as necessary to handle these sub-expressions.

Output

The result should be a single string value that is returned back to the **`main`** function provided for Project 4 (do **not** change this **`main`**; it will be used by the grader). This string should be a valid C program that can be compiled (by **`gcc`** on **`stdsun`**) and executed. This output program should be a correct translation of the original program. The only statements allowed in the output program are the ones described in the three-address code definition from the class notes:

- Assignments: **`x = y op z; x = op y; x = y; x = y[z]; x[y] = z;`** as discussed in class
- Conditional jumps: **`if (x relop y) goto L;`**
- Unconditional jumps: **`goto L;`**
- Returns: **`return x;`** where **`x`** is a program variable, a temporary variable, or a constant
- Null statements (Section 6.8.3): just a semicolon, as in **`_l2: ;`** from the earlier example
- Each of the above statements could be labeled or unlabeled (but null statements should always be labeled). Note that a labeled statement could itself be labeled with another label.

Of course, the output program will also contain the variable declarations from the input program as well as the declarations of all necessary temporary variables. Note that we do **not** allow block statements in the output program: inside the function body, there should not be block statements **`{…}`**. You can assume that the elimination of blocks will **not** create any name conflicts – i.e., there will not be variables with the same name that exist in different scopes in the original program.

Details

- Your submission must compile and run on **`stdlinux`**, using the ROSE installation in **`/class/cse756/…`**
- During the analysis of the AST, do **not** print directly to **`cout`**. The only printing to **`cout`** must be done by the provided **`main`**.
- Do **not** use the ROSE methods **`unparseToString`** and **`unparseToCompleteString`**.
- Your submission must work correctly on the modfft1.c test program used for Projects 2, 3, and 4.
- You can name your labels **`_l1`**, **`_l2`**, **`_l3`**, etc. Assume that the input program does not already use such names.
- You do **not** need to transform the expressions in variable declarations (e.g., **`float w=x+y+z;`** should remain the same).
- You **do** need to transform the initialization expressions and increment expressions for loops (e.g., all three components of **`for(w=x+y+z;w<p+q+r;w+=a+b+c)`** should be transformed).

Project Submission

On or before 11:59 pm on the due date, you should submit a single file intermediate2.cpp containing all of your code; it should work with **`main`** from Project 4. Submit your project using "**`submit c756aa lab5 intermediate2.cpp`**".

If the time stamp on your electronic submission is **12:00 am on the next day or later**, you will receive 10% reduction per day, for up to three days. If your submission is later than 3 days after the deadline, it will not be accepted and you will receive zero points for this project. If you resubmit your project, this will override any previous submissions and only **the latest** submission will be considered – **resubmit at your own risk**.

Academic Integrity

The project you submit must be your own work. Minor consultations with others in the class are OK. The work on the project should be your own: all the design, programming, and testing should be done independently. Submissions that show excessive similarities will be taken as evidence of cheating and dealt with accordingly.