CSE 756, Project 4: Intermediate Code for Expressions

You need to extend your solution for Project 3 to translate Sage AST expressions to intermediate code. As with Project 3, 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 3; 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 8 (Tuesday)**, 11:59 pm.

Input Language

Assume that the input is a C program that satisfies all restrictions described in Project 3, with the following additional restrictions on all *expression statements* (defined in Section 6.8.3 of the ANSI C document; ignore null expressions) and *return statements* (defined in Section 6.8.6):

- The expressions do not contain binary operators **>**, **>=**, **<**, **<=**.
- The expressions do not contain unary operator ***** (pointer dereference). Furthermore, you can assume that the program does not contain any pointer-typed variables at all. Note that this restriction eliminates the use of modfft2.c as a test program, since it contains both pointer-typed variables and pointer dereference expressions.
- There are no multi-dimensional arrays: every array is one-dimensional.

Furthermore, assume that all variable declarations for local variables are in the outermost scope in the function body – that is, block statements inside the function body do not contain variable declarations.

Translation

Consider every *expression statement* in the input program. In the Sage AST in ROSE, all such statements are represented by SgExprStatement nodes. 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 program is

```
main(){
      float a = 1, b = 2, c = 3, d;
      d = a + b + c;
}
```

the resulting program could be something like

```
main(){
      float _t1, _t2, …;
      float a = 1, b = 2, c = 3, d;
      _t1 = a;
      _t2 = b; …
      d = …;
}
```

Note that you need to create the temporary variables as regular C variables, with the appropriate declarations. The location of these declarations in the output program is irrelevant (e.g., they could all come at the very beginning of the function, before anything else), as long as the output is a valid C program that is a correct translation of the input program.

You also need to translate all *return statements*: the expression in **`return`** ***expression ;*** should be simplified to **`return`** ***x ;*** where ***x*** is a single address (i.e., a program variable, a temporary variable, or a constant) corresponding to *expression.addr* from the syntax-directed definition (SDD) discussed in class.

Output

The result should be a single string value that is returned back to the **`main`** function provided for Project 3 (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, based on the SDD we discussed.

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 **`main`**.
- Do **not** use the ROSE methods **`unparseToString`** and **`unparseToCompleteString`**.
- Your submission must work correctly on the modfft1.c test program used for Project 2 and Project 3. As mentioned earlier, modfft2.c is not a valid test program due to the use of pointer-typed variables.
- You can name your temporary variables **`_t1`**, **`_t2`**, **`_t3`**, etc. Assume that the input program does not already use such names.
- You need to transform **only** expression statements and return statements. For everything else in the input program, just use your printing code from Project 3. For example, you should **not** modify in any way the expressions in variable declarations – e.g., **`float w=x+y+z;`** should remain the same. Similarly, you should **not** modify the expressions in loops and ifs – e.g., **`for(w=x+y+z;w<p+q+r;w+=a+b+c)`** should remain the same. Note that the conditions of loops/if and the initializations of loops (e.g., **`w=x+y+z`** and **`w<p+q+r`** above) are represented in the Sage AST as SgExprStatement nodes. Unlike the "normal" SgExprStatement nodes for actual expression statements, these nodes should **not** be translated to three-address code.
- Note that the input program could contain implicit casts. For example, in the expression statement **`data_real[i]=0.33*i;`** in modfft1.c, the array elements are of type **`double`**, but the variable **`i`** is of type **`int`** and its value is implicitly casted to **`double`**. You do **not** need to create explicit casts in the output three-address code. Assume that the implicit casts in your three-address code will work correctly – e.g., it is OK to create **`_t1=0.33*_t2`** where one temporary variable is of type **`double`**, but the other temporary is of type **`int`**.

Project Submission

On or before 11:59 pm on the due date, you should submit a single file intermediate1.cpp containing all of your code; it should work with **`main`** from Project 3. Submit your project using "**`submit c756aa lab4 intermediate1.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.