Compiler Theory CS447b: Quiz 4. **UWO, April 11, 2003.**

Student ID number:

Guidelines. The quiz is closed book and all notes are forbidden. The duration of the quiz is 30 minutes. All answers should be written in the *answer boxes*. No justifications for the answers are needed. You are expected to do this quiz on your own without assistance from anyone else in the class. If possible, please avoid pencils and use pens with dark ink. Thank you.

Exercise 1. Consider the following three-address code fragment.

```
    a := 1
    b := 2
L2  c := a + b
    d := c - a
    if d = b goto L5
L3  d := b * d
    if d = b goto L5
    d := a + b
    e := e + 1
    goto L3
L5  b := a + b
    e := c - a
    if d = b goto L2
    a := b + d
    b := a - d
```

1. Give the flow graph of this program.
2. Identify the back edges and the loops.

Note that the results of these computations do not matter. We are only interested here in the relations between the basic blocks.

Answer 1

Figure 1 shows the control flow graph G of the above TAC fragment.
In the table below, we give the dominators of each basic block.

Basic block	Dominators
B_1	B_1
B_2	B_1, B_2
B_3	B_1, B_2, B_3
B_4	B_1, B_2, B_3, B_4
B_5	B_1, B_2, B_5
B_6	B_1, B_2, B_5, B_6

This shows that there are two (and only two) back edges (i.e. an edge from a point to a dominator of this point) namely: (B_4, B_3) and (B_5, B_2). Therefore we expect to find at least two loops: one based on the back edge (B_4, B_3) and one based on (B_5, B_2). As we shall see now, one of these loops involve the two back edges and the second loop is nested in the first one.
The subgraph consisting of

- the nodes B_2, B_3, B_4, B_5 and
- the six edges of G connecting them

is a loop G_1 since

- it is strongly connected (there is path from any of this node to any other)
- there is a unique entry point ($B2$) from the rest of G.

Observe that

- the subgraph consisting of the nodes B_3, B_4 and the two edges connecting them is **also** a loop G_2, nested in G_1,
- the subgraph consisting of the nodes B_2, B_5 and the two edges connecting them is **not** a loop since it has two entry points.

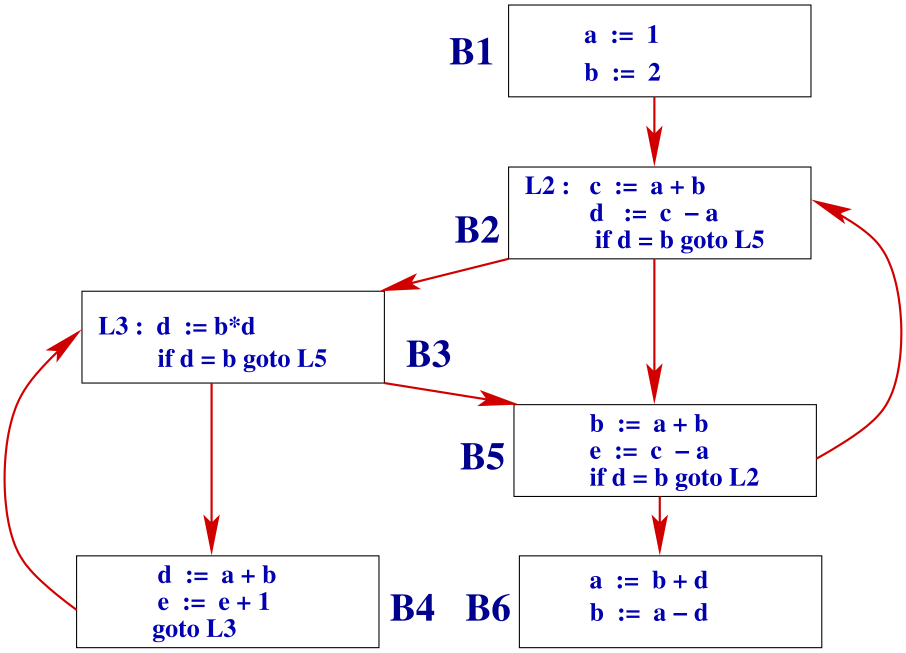

Figure 1: The control flow graph.

.

Exercise 2. The grammar below generates sequences of declarations of the form $\mathbf{id} : T$.

$$
\begin{array}{lcl}
P & \longmapsto & D \\
D & \longmapsto & D \,;\, D \\
D & \longmapsto & \mathbf{id} : T \\
T & \longmapsto & \mathbf{integer} \\
T & \longmapsto & \mathbf{float} \\
T & \longmapsto & \mathbf{array}\ [\ \mathbf{num}\ ]\ \mathbf{of}\ T_1 \\
T & \longmapsto & \uparrow T_1
\end{array}
$$

As usual **id** is the token for identifiers, **integer** denotes a type of integers, **float** denotes a type of floating point numbers and **num** is the token for integer values. Note that **array**, $\uparrow$, **;** , **:**, **[**, **]** and **of** are terminals too. Moreover,

- $\mathbf{array}\ [\ \mathbf{num}\ ]\ \mathbf{of}\ T$ denotes the type of arrays of length $\mathbf{num}.val$ with elements of type $T.type$.
- $\uparrow T$ denotes the type of pointers to elements of type $T.type$.

You are asked to write a **translation scheme** which computes the memory address of every declared name and stores this information in the symbol table. To reach that goal you may associate the nonterminal T with a synthesized attribute $T.width$ that indicates the number of memory units taken by objects of type T.

- For $T \longmapsto \mathbf{integer}$ and $T \longmapsto \mathbf{float}$ it is usual to have $T.width = 4$ and $T.width = 8$ respectively.
- For $T \longmapsto \mathbf{array}\ [\ \mathbf{num}\ ]\ \mathbf{of}\ T$ the formula for $T.width$ is easy to guess.
- For $T \longmapsto \uparrow T_1$ remember that a pointer is an integer.

As usual, the nonterminal T has a synthesized attribute $T.type$. You are **not** asked to give the semantic rules computing $T.type$ (although this is easy). You may need a global variable $offset$ which gives the current available memory address:

- Before the first declaration is made, $offset$ should be initialized to 0.
- As each name $\mathbf{id}.name$ is seen, that name is entered in the symbol table with $offset$ equal to the current value and $offset$ is incremented by the width of the type of $\mathbf{id}.name$.
- The procedure call $\mathsf{enter}(\mathbf{id}.name, T.type, offset)$ creates a symbol-table entry for $\mathbf{id}.name$, with type $T.type$ and memory address $offset$.

Answer 2

Observe that $offset$ must be initialized before any other action is executed.

P	$\longmapsto$		$\{offset := 0\}$
		D	
D	$\longmapsto$	D **;** D	
D	$\longmapsto$	**id** : T	$\{$ enter($\mathbf{id}.name, T.type, offset$) $offset := offset + T.with\}$
T	$\longmapsto$	**integer**	$\{T.with := 4\}$
T	$\longmapsto$	**real**	$\{T.with := 8\}$
T	$\longmapsto$	**array** [**num**] **of** T_1	$\{T.with := \mathbf{num}.val \ \times \ T.with\}$
T	$\longmapsto$	$\uparrow T_1$	$\{T.with := 4\}$

Exercise 3. Translate the following C-like piece of code into three-address code.

```
a = x * y + u * v - w * t;
b = w * t - m * n;
while (b != 0) {
   tmp = a;
   a = b;
   while (tmp > b) {
      tmp = tmp - b;
   }
   b = tmp;
}
```

The three-address code instructions you may need are of one of the forms:

- `goto L` where is a symbolic label (like `L1, L2, ...`)
- `if x relop y goto L` where `relop` can be = or <
- `x:= y op z` where `op` can be −, − and `*`

Moreover your solution does **not** need to be obtained by one of the algorithms of the course. Just consider that you are programming a very basic calculator.

Answer 3

```
      tmp := w * t
      a := - tmp
      b := tmp
      tmp := x * y
      a := a + tmp
      tmp := u * v
      a := a + tmp
      tmp := m * n
      b := b - tmp
L1    if b = 0 goto L4
      tmp := a
      a := b
L2    if tmp < b goto L3
      if tmp = b goto L3
      tmp := tmp - b
      goto L2
L3    b := tmp
      goto L1
L4
```