| Source WHILE | Code intermédiaire |
|---|---|
| if E1 then<br>    C1<br>else<br>    C2<br>fi | { E.place = ε<br>  E.true = newLabel();<br>  E.false = newLabel();<br>  E.end = newLabel();<br>  E.code = E1.code(E.true, E.false) ·<br>           <label E.true, _, _, _> ·<br>           C1.code ·<br>           <goto E.end, _, _, _> ·<br>           <label E.false, _, _, _> ·<br>           C2.code ·<br>           <label E.end, _, _, _> } |
| while E1 do<br>    C<br>od | { E.place = ε<br>  E.body = newLabel();<br>  E.ifFalse = newLabel();<br>  E.cond = newLabel();<br>  E.code = <label E.cond, _, _, _> ·<br>           E1.code(E.body, E.ifFalse) ·<br>           <label E.body, _, _, _> ·<br>           C.code ·<br>           <goto E.cond, _, _, _> ·<br>           <label E.ifFalse, _, _, _> } |
| nop | { E.place = ε<br>  E.code = <nop, _, _, _> } |
| V1,Vn := E1,En | { E.place = ε<br>  E.code = E1.code ·<br>           <write, _, E1.place, _> ·<br>           En.code ·<br>           <write, _, En.place, _> ·<br>           <read, Vn, _, _> ·<br>           <read, V1, _, _> } |
| for E1 do<br>    C<br>od | { E.place = newVariable();<br>  E.body = newLabel();<br>  E.end = newLabel();<br>  E.code = <write, _, E1.place, _> ·<br>           <read, E.place, _, _> ·<br>           E1.code ·<br>           <label E.body, _, _, _> ·<br>           <iff E.end, _, E.place, _> ·<br>           C.code ·<br>           <tl, E.place, E.place, _> ·<br>           <goto E.body, _, _, _> }<br>           <label E.end, _, _, _> } |

| | |
|---|---|
| `foreach V1 in E1 do`<br>`    C`<br>`od` | `{ E.place = newVariable();`<br>`  E.exp = TS(V1);`<br>`  E.body = newLabel();`<br>`  E.end = newLabel();`<br>`  E.code = <write, _, E1.place, _> ·`<br>`           <read, E.exp, _, _> ·`<br>`           E1.code ·`<br>`           <label E.body, _, _, _> ·`<br>`           <iff E.end, _, E.exp, _> ·`<br>`           C.code ·`<br>`           <hd, E.place, E.exp, _> ·`<br>`           <tl, E.exp, E.exp, _> ·`<br>`           <goto E.body, _, _, _> }`<br>`           <label E.end, _, _, _> }` |
| `E1 and … and En` | `{ E.place = newVariable();`<br>`  E.ifTrue = newLabel();`<br>`  E.ifFalse = newLabel();`<br>`  E.code = E1.code(E.ifTrue, E.ifFalse) ·`<br>`           <iff E.ifFalse, _, E1.place, _> ·`<br>`           En.code(E.ifTrue, E.ifFalse) ·`<br>`           <iff E.ifFalse, _, En.place, _> ·`<br>`           <label E.ifTrue, _, _, _> ·`<br>`           <true, E.place, _, _> ·`<br>`           <goto E.true, _, _, _> ·`<br>`           <label E.ifFalse, _, _, _> ·`<br>`           <false, E.place, _, _> ·`<br>`           <goto E.false, _, _, _> }` |
| `E1 or … or En` | `{ E.place = newVariable();`<br>`  E.ifTrue = newLabel();`<br>`  E.ifFalse = newLabel();`<br>`  E.end = newLabel();`<br>`  E.code = E1.code(E.ifTrue, E.ifFalse) ·`<br>`           <ift E.ifTrue, _, E1.place, _> ·`<br>`           En.code(E.ifTrue, E.ifFalse) ·`<br>`           <ift E.ifTrue, _, En.place, _> ·`<br>`           <goto E.ifFalse, _, _, _> ·`<br>`           <label E.ifTrue, _, _, _> ·`<br>`           <true, E.place, _, _> ·`<br>`           <goto E.end, _, _, _> ·`<br>`           <label E.ifFalse, _, _, _> ·`<br>`           <false, E.place, _, _> ·`<br>`           <label E.end, _, _, _> }` |
| `not E1` | `{ E.place = newVariable();`<br>`  E.ifTrue = newLabel();`<br>`  E.end = newLabel();` |

| | |
|---|---|
| | E.code = E1.code(E.true, E.false) ·<br>         <ift E.ifTrue, _, E1.place, _> ·<br>         <true, E.place, _, _> ·<br>         <goto E.end, _, _, _> ·<br>         <label E.ifTrue, _, _, _> ·<br>         <false, E.place, _, _> ·<br>         <label E.end, _, _, _> } |
| E1 =? E2 | { E.place = newVariable(); // Modifié à l’éxecution<br>  E.ifEqual = newLabel();<br>  E.end = newLabel();<br>  E.code = E1.code(E.true, E.false) ·<br>         E2.code(E.true, E.false) ·<br>         <ifeq E.ifEqual,_, E1.place, E2.place> ·<br>         <false, E.place, _, _> ·<br>         <goto E.end, _, _, _> ·<br>         <label E.ifEqual, _, _, _> ·<br>         <true, E.place, _, _> ·<br>         <label E.end, _, _, _> } |
| E → nil | { E.place = newVariable();<br>  E.code = <nil, E.place, _, _> } |
| E → symb | { E.place = TS(symb);<br>  E.code = ε } |
| E → var | { E.place = TS(var);<br>  Si expression booléenne :<br>  E.code = <ift E.true, _, E.place, _> ·<br>         <goto E.false, _, _, _><br>  Sinon :<br>  E.code = ε } |
| cons E1 … En | Si n = 1 :<br>{ E.code = E1.code }<br>Sinon si n >= 2 :<br>{ E.place = newVariable();<br>  E.code = E1.code() ·<br>         E2.code() ·<br>         <cons, E.place, E2.place, E1.place> ·<br>         En.code() ·<br>         <cons, E.place, En.place, E.place> } |
| list E1 … En | { E.place = newVariable();<br>  E.nil = newVariable();<br>  E.code = <nil, E.nil, _, _><br>         E1.code() ·<br>         <cons, E.place, E1.place, E.nil> ·<br>         En.code() · |

| | |
|---|---|
| | `<cons, E.place, En.place, E.place> }` |
| `hd` **`E1`** | `{` **`E.place`** `= newVariable();` **`E.code`** `= E1.code() • <hd, E.place, E1.place, _> }` |
| `tl` **`E1`** | `{` **`E.place`** `= newVariable();` **`E.code`** `= E1.code() • <tl, E.place, E1.place, _> }` |
| `call` **`func A1 An`** | `{` **`E.place`** `= newVariable();` **`E.code`** `= A1.code() • An.code() • <write, _, An.place, _> • <write, _, A1.place, _> • <call func, _, _, _> • <read, E.place, _, _> }` |

| Code intermédiaire | Code C++ |
|---|---|
| <nop, _, _, _> | bin_tree::nop(); |
| <symb S1, D, _, _> | D = make_shared<bin_tree>(S1); |
| <read, D, _, _> | D = f_stack.top(); f_stack.pop(); |
| <write, _, A1, _> | f_stack.push(A1); |
| <goto LAB, _, _, _> | goto LAB; |
| <label LAB, _, _, _> | LAB : |
| <ifeq LAB, _, A1, A2> | if(bin_tree::equals(A1, A2)) { goto LAB; } |
| <ift LAB, _, A1, _> | if(A1->isTrue()) { goto LAB; } |
| <iff LAB, _, A1, _> | if(A1->isFalse()) { goto LAB; } |
| <true, D, _, _> | D = bin_tree::getTrue(); |
| <false, D, _, _> | D = bin_tree::getFalse(); |
| <nil, D, _, _> | D = bin_tree::nil(); |
| <cons, D, A1, A2> | D = bin_tree::cons(A1, A2); |
| <hd, D, A1, _> | D = bin_tree::hd(A1); |
| <tl, D, A1, _> | D = bin_tree::tl(A1); |
| <call func, _, _, _> | func(f_stack); |