CISC 858: Team “iCompile”:

Phase 4 Test Cases

Mike Michaud

Tammy Jiang

1.Test Cases:

In this phase we tested our changes to Code Generation, as part of changes to support the new JT language. This document contains the test cases required to test the new JT Grammar. Each test case indicates the test input file and the output produced.

2.Switch & If-elsif-else Statements:

Source Code: testSwitch.pt
Assembly Code: testSwitch.s
Executable Output: testSwitch.out

Test Purpose:
In this test, we covered the following:

1. Switch statements choose the right case.
2. if-elsif-if statements choose the right case.

testSwitch.pt	**Execution of testSwitch.out**

`program testSwitch(output) {`
`    var a : integer;`
`    var b : integer;`
`    a = 1;`
`    switch(a)`
`    {`
`        case 1: case 2: case 3:`
`            b = b + 2;`
`            write("I am in case 1, 2, 3");`
`            break;`
`        case 4:`
`            b = b + 1;`
`            write("I am in case 4");`
`            break;`
`    }`
`    writeln();`
	
	`// a == 1`
	
	`I am in case 1, 2, 3`

<table>
<tr>
<td>

```
    a = 2;
    if(a == 1)
        write("a equals to 1");
    elsif(a == 2)
        write("a equals to 2");
    else
        write("a neither equals to 1
or 2");

    writeln();

    if(a != 1)
        write("a doesn't equal to
1");
    else
        write("a equals to 1");
    writeln();
}
```

</td>
<td>

```

// a == 2 now
a equals to 2








// a != 1 is true
a doesn't equal to 1
```

</td>
</tr>
</table>

3.Loop-break when Statements:

Source Code: testLoopBreakWhen.pt
Assembly Code: testLoopBreakWhen.s
Executable Output: testLoopBreakWhen.out

Test Purpose:

In this test, we covered the following:

1. Loop-break-when statements handled properly.

<table>
<tr>
<th>testLoopBreakWhen.pt</th>
<th>Execution of testLoopBreakWhen.out</th>
</tr>
<tr>
<td>

```
// print a triangle of stars
program P (output) {

    var stars: string;
    var i: integer;

    i = 1;
    stars = "a";
    loop {
        break when (i > 3);
        stars = "*" + stars;
        write (stars); writeln ();
        i = i + 1;
    }
}
```

</td>
<td>

```






// it has three rounds of execution.

*a

**a

***a
```

</td>
</tr>
</table>

4.String Operations:

Source Code: testStringOperations.pt

Assembly Code: testStringOperations.s
Executable Output: testStringOperations.out

Test Purpose:

In this test, we covered the following:

1. String concatenation using overloaded ‘+’ operator.
2. String Length using unary operator ‘#’.
3. Substring using ternary operator ‘@’.
4. String equality using “==”.

testStringOperations.pt	**Execution of testStringOperations.out**

`program Hello (output)`
`{`
`var string1: string;`
`var string2: string;`
`var string3: string;`
`var i: integer;`
`var j : integer;`
`var tf: boolean;`	

`string1 = "Hello ";`
`string2 = "World";`	

`string3 = string1 + string2;`
`write( "concatenate them: ", string3 );`
`writeln();`
	`// string concatenation.`
	`concatenate them: Hello World`

`string3 = string1 @ 2 : 4;`
`write( "Then (substring): ", string3 );`
`writeln();`
	`// substring`
	`Then (substring): ell`

`//String Length output as integer, which currently has error`
`i = #string3 ;`
`write( "Then length is: ", i );`
`writeln();`
`if(#"a" == 1)`
`write("The length is 1");`
`else`
`write("The length is not 1");`
`writeln();`
	`// string length`
	`Then length is:          3`
	`The length is 1`

`if ( string3 == string1 )`
`write( "The strings are Equal!" );`
`else`
`write( "Then strings are NOT Equal." );`
`writeln();`
`}`
	`// compare strings.`
	`Then strings are NOT Equal.`

4.Empty String:

Source Code: testEmptyString.pt
Assembly Code: testEmptyString.s
Executable Output: testEmptyString.out

Test Purpose:
In this test, we covered the following:

1. Support for empty strings.

testEmptyString.pt	Execution of testEmptyString.out

program a(output) {
var a : string;
a = "";
write(a);
write("abc");
write(" ");
}
	abc **<space>**

6.Function Return type:

Source Code: testFunction.pt
Assembly Code: testFunction.s
Executable Output: testFunction.out

Test Purpose:
In this test, we covered the following:

1. User-defined function calls are accepted.
2. User-defined function returns string type (pass by reference).
3. User-defined function returns integer type (pass by value).

testFunction.pt	Execution of testFunction.out

program primes (output) {
function test1 (var f: string):string {
var a : string;
a = f + "2";
return (a);
}
 	
function test2(k : integer): integer {
var t : integer;
t = k + 1;
return (t);
}
	// this function returns any string concatenate “2”.
	
	// this function return any input integers plus 1.

```
      var i : string;
      var j : string;
      var t : integer;

      i = "test";

      j = test1(i);
      write(j);                         test2
      writeln();

      t = test2(1);
      write(t);                         2
      writeln();
}
```

7.Class Statements:

Source Code: testClass.pt
Assembly Code: testClass.s
Executable Output: testClass.out

Test Purpose:
In this test, we covered the following:

1. Statements inside class.
2. Function calls and Function Declarations inside class.

testClass.pt | **Execution of testClass.out**

```
program testclass(output)
{
      let maxVal = 99;
    class simpleClass {
            var list : array [1 ..
10] of integer;
            var a : integer;
            public function swap(i :
integer, j : integer) {
                  var tmp : integer;
                  tmp = i;
                  i = j;
                  j = tmp;
            }

            public function test( a :
integer) {
                  write(a);
                  writeln();
            }

            list[1] = 61;
```

list[2] = 61;
    }
 	
      test(3);
      writeln();
}
	
	3

***NOTE: in our implementation of classes, there is currently one bug which generates an Assertion #25. If we declare a function or procedure after a class declaration, the code generator generates a complete “*.s” file, but no executable file. The following “.pt” file is the test file we used:

```
program P (output) {

      let nelements = 25;
      let maxval = 99;

    class elements {
      var list : array [1 .. nelements] of integer;
      var current : integer;
      var i,j: integer;

      public function lessthan (i : integer, j : integer, var b : boolean)
      {
            current = i;
            b = list [i] < list [j];
      }

      public function swap (i: integer, j : integer)
      {
            var t : integer;
            current = j;
            t = list [i];
            list [i] = list [j];
            list [j] = t;
      }

      public function print ()
      {
            var i: integer;
            i = 1;
            loop {
                  write (list[i]:4);
                  i = i + 1;
                  break when (i > nelements);
            }
            writeln ();
      }

      list[1] = 61;
      }

      function sort ()
      {
      var i,j: integer;
      var lt: boolean;
      i = nelements - 1;
      loop {
            break when (i <= 0);
            j = 1;
            loop {
                  break when (j >= nelements);
                  lessthan (j, j+1, lt);
                  if (not lt)
                  swap (j, j + 1);
                  j = j + 1;
            }
            i = i - 1;
      }

    sort();
}
```