COMPILADORES
REPORTE #2
HOC5 -- Condicionales y Ciclos

21 de Noviembre del 2018

PROFESOR: TECLA PARRA ROBERTO

ALUMNO: SALDAÑA AGUILAR ANDRÉS

GRUPO: 3CM8

Introducción

Los **ciclos while** son también una estructura cíclica, que nos permite ejecutar una o varias líneas de código de manera repetitiva sin necesidad de tener un valor inicial e incluso a veces sin siquiera conocer cuando se va a dar el valor final que esperamos, los ciclos while, no dependen directamente de valores numéricos, sino de valores booleanos, es decir su ejecución depende del valor de verdad de una condición dada, verdadera o falso, nada más. De este modo los ciclos while, son mucho más efectivos para condiciones indeterminadas, que no conocemos cuándo se van a dar a diferencia de los ciclos for, con los cuales se debe tener claro un principio, un final y un tamaño de paso.

La **condición**, encerrada entre paréntesis, es una expresión que puede dar como resultado 0 (interpretado como falso) o cualquier valor distinto de 0 (interpretado como verdadero). Cuando la condición sea verdadera, se ejecutarán las sentencias dentro del primer bloque de código, cuando la condición sea falsa, se ejecutarán las sentencias del segundo bloque de código. Las expresiones y valores de tipo verdadero/falso son también llamados valores lógicos o booleanos.

Desarrollo

HOC5 tiene como objetivo poder realizar ciclos y condicionales con complejos como constantes o variables, esto es posible extendiendo la gramática del programa de HOC4, añadiendo las siguientes reglas:

```
stmt: exp { maq.code("pop"); }
    | PRINT exp { maq.code("printComplex");
                  $$ = new ParserVal($2.obj);
                }
    | while cond stmt end { maq.getProg().setElementAt($3.obj, (int) $1.obj + 1);
                            maq.getProg().setElementAt($4.obj, (int) $1.obj + 2);
                          }
    | if cond stmt end    { maq.getProg().setElementAt($3.obj, (int) $1.obj + 1);
                            maq.getProg().setElementAt($4.obj, (int) $1.obj + 3);
                          }
    | if cond stmt end ELSE stmt end {  maq.getProg().setElementAt($3.obj, (int) $1.obj +
1);
                                        maq.getProg().setElementAt($6.obj, (int) $1.obj + 2);
                                        maq.getProg().setElementAt($7.obj, (int) $1.obj + 3);
```

```
                                }
    | '{' stmtlist '}' {$$  =  $2; }
    ;
cond: '(' exp ')' { maq.code("STOP");
                $$ = new ParserVal($2.obj);
              }
     ;
```

Donde en las reglas de producción, tienen en común el uso del método **getProg( ),** que se encarga de conseguir la memoria del programa (RAM) con el objetivo de que podamos colocar instrucciones en posiciones específicas haciendo uso del método **setElementAt( )** para poder simular asi el funcionamiento de un ciclo o condición.
También es importante ver el contenido de una condición o ciclo como una lista de expresiones a ejecutar si es que la condición de entrada se cumple.
Después, tenemos las reglas de producción para un if o un while, que son terminales y tienen como propósito guardar la instrucción que contiene la lógica de ejecución de la condición o ciclo:

```
while:  WHILE       { int numI = maq.code("whileCode");
                      maq.code("STOP"); maq.code("STOP");
                      $$ = new ParserVal(new Integer(numI));
                    }
    ;
if: IF  { int numI = maq.code("ifCode");
        maq.code("STOP"); maq.code("STOP"); maq.code("STOP");
        $$ = new ParserVal(new Integer(numI));
      }
    ;
```

Estas instrucciónes se instalan en la memoria del programa como cadenas que serán posteriormente llamadas y ejecutadas por la máquina virtual

```
 /* Code of while */
```

```
void whileCode(){
  boolean d;
  int savepc = pc;
  execute(savepc + 2);  /* condition */
  d = ((Boolean)pila.pop()).booleanValue();

  while (d) {
    execute(((Integer)prog.elementAt(savepc)).intValue()); // statement
    execute(savepc + 2); // condition
    d = ((Boolean)pila.pop()).booleanValue();
  }
  pc = ((Integer)prog.elementAt(savepc + 1)).intValue();
 }
 /* Code of if */
void ifCode(){
  boolean d;
  int savepc = pc;
  execute(savepc + 3); /* Condition */
  d=((Boolean)pila.pop()).booleanValue();
  if (d) {
    execute(((Integer)prog.elementAt(savepc)).intValue()); /* stm1 */
  } else if (!prog.elementAt(savepc + 1).toString().equals("STOP")) {
    execute(((Integer)prog.elementAt(savepc + 1)).intValue()); /* stm2 */
  }
  pc = ((Integer)prog.elementAt(savepc + 2)).intValue();
```

En las funciones conseguimos el resultado de la evaluación de una expresión que da como resultado un booleano con el método pop( ) y ocupamos condicionales o ciclos ya implementados por Java para simular el comportamiento de un if, else o while.

Resultados:

Para ejemplificar el funcionamiento del programa, ingrese el código para elevar un número a la novena potencia, haciendo uso de dos variables:

numa: encargada de funcionar como contador para el ciclo while

numb: encargada de multiplicarse a sí misma “n” veces

```
.:: Complex Number Calculator ::.
Expression: numa = 1+0i
Expression: numb = 1+1i
Expression: while(numa < 10+10i){numb = numb*1+1i numa=numa+1+0i print(numb) }
Result:
0.0 + 2.0 i
Result:
-2.0 + 2.0 i
Result:
-4.0 + 0.0 i
Result:
-4.0 -4.0 i
Result:
0.0 -8.0 i
Result:
8.0 -8.0 i
Result:
16.0 + 0.0 i
Result:
16.0 + 16.0 i
Result:
0.0 + 32.0 i
Expression:
```

Ahora, imprimimos en pantalla el resultado de la comparación de numa (0+32i) con numb (10+0i), que en este caso caerá en el else imprimiendo numb:

```
Expression: if( numa > numb ){ print(numa) } else{ print(numb) }
Result:
10.0 + 0.0 i
Expression:
```

Conclusión

En esta práctica pusimos en práctica como manipular la memoria del programa y la ejecución de sus instrucciones para realizar funciones como if, else y while, estuvo muy interesante y tomó su tiempo entenderlo.