



# INSTITUTO POLITECNICO NACIONAL

## ESCUELA SUPERIOR DE CÓMPUTO (ESCOM)



### COMPILADORES

NOMBRE DEL ALUMNO:

- SANTOS MÉNDEZ ULISES JESÚS

PRÁCTICA:

- DECISIONES Y CICLOS

NÚMERO DE PRÁCTICA: 5

FECHA DE ENTREGA:

- 09/06/2023

GRUPO:

- 3CM14

## Calculadora de vectores con condicionales y ciclos

## Introducción:

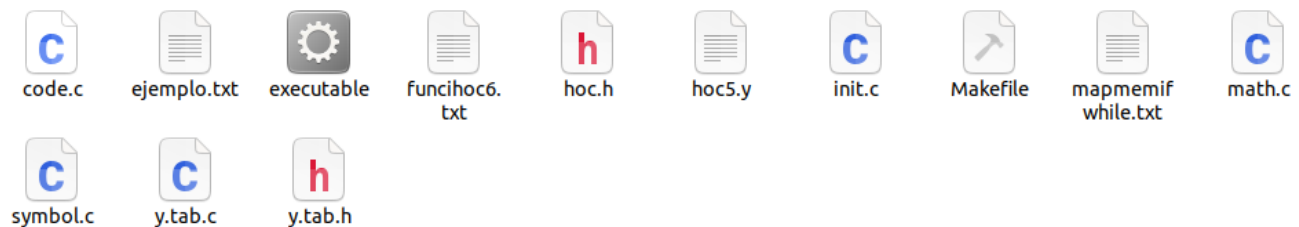
En esta práctica se busca aplicar lo que se aprende durante la clase, respecto a la implementación de condicionales y ciclos haciendo uso de la práctica anterior y los documentos que se tienen del HOC5, se hace uso del YACC en C y C++ que se utilizó en la práctica 1 y en la práctica 4, se harán algunas modificaciones necesarias relacionadas con lo que el maestro compartió, todo esto va en relación que se compiló y se ejecutó el HOC 5.

### Objetivos:

- Modificar la especificación de YACC de HOC5 en la práctica 4 y agregando una parte dval de tipo double a la unión Datum en el archivo hoc.h para almacenar el resultado de las operaciones de comparación.
- Se busca el implementar operadores relacionales para poder realizar comparaciones y ver si las condiciones se cumplen como lo son gt,lt,eq,ge,le,ne.
- Se deben de modificar las funciones de ifcode, whilecode que se encuentran en el archivo code.c para que trabajen con la parte dval de tipo double de la unión Datum.

### Desarrollo:

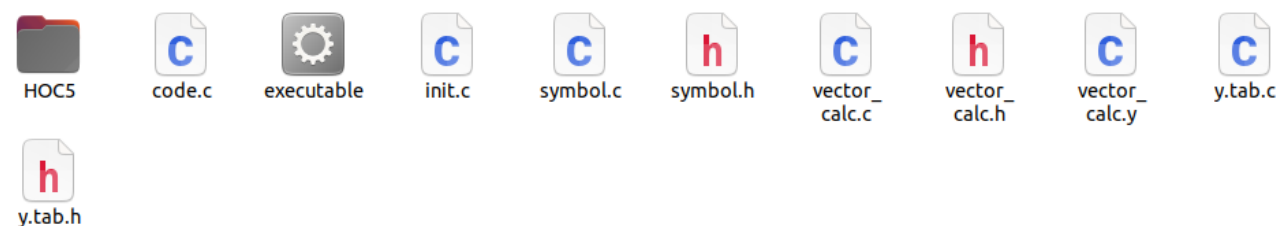
Principalmente se revisa la carpeta de HOC5 y se tienen los siguientes archivos:



Se abre una terminal donde se compilo el archivo hoc5.y y se compilaron los archivos en C

```
ulisespc04@ulisesm04:/media/ulisespc04/Ulises/OCTAVO SEMESTRE/Compiladores/Prac
ticas/Practica 5/HOCS$ ./executable
yyy= 100
while (yyy < 1000){print yyy yyy=yyy+10 }
100 110 120 130 140 150 160 170 180 190 200 210 220 230 240 250 260 270 280 290 300 310 320 330 340 350 360 370 380 390 400 410 420
430 440 450 460 470 480 490 500 510 520 530 540 550 560 570 580 590 600 610 620 630 640 650 660 670 680 690 700 710 720 730 740 750
760 770 780 790 800 810 820 830 840 850 860 870 880 890 900 910 920 930 940 950 960 970 980 990
```

Ahora se copiaron archivos de la práctica 4 de la calculadora de vectores y se implementaron las condicionales y ciclos, los archivos en conjunto son:



En la gramática de nuestra calculadora de vectores tenemos la definición de nuevos tokens así como de otras producciones que nos van a permitir construir la condicional y agregar el ciclo while como se muestra en la imagen a continuación:

```
stmt:  asgnEscalar          { code(pop1); }
      | asgnVector          { code(pop1); }
      | PRINT expEscalar    { code(prexpresc); $$ = $2; }
      | PRINT expVectorial  { code(prexpvec); $$ = $2; }
      | while cond stmt end { ($1)[1] = (Inst)$3; /* (
      |                       | ($1)[2] = (Inst)$4; } /* (
      | if cond stmt end     { /* proposicion if que no
      |                       | ($1)[1] = (Inst)$3; /*
      |                       | ($1)[3] = (Inst)$4;
      |                       } /* terminar si la condicio
      | if cond stmt end ELSE stmt end { /* proposicion if con parte
      |                               | ($1)[1] = (Inst)$3; /*
      |                               | ($1)[2] = (Inst)$6; /*
      |                               | ($1)[3] = (Inst)$7; }
      | '{' stmtlist '}'      { $$ = $2; }
      ;
cond:  '(' expEscalar ')'    { code(STOP); $$ = $2; }
      ;
while: WHILE { $$ = code3(whilecode,STOP,STOP); }
      ;
if:    IF { $$=code(ifcode); code3(STOP, STOP, STOP); }
      ;
end:   /* nada */{ code(STOP); $$ = prog; }
```

También se agregaron los operadores relacionales que permitirán realizar la comparación entre valores:

```
//ESCALAR-ESCALAR
| expEscalar GT expEscalar { code(gtEE); }
| expEscalar GE expEscalar { code(geEE); }
| expEscalar LT expEscalar { code(ltEE); }
| expEscalar LE expEscalar { code(leEE); }
| expEscalar EQ expEscalar { code(eqEE); }
| expEscalar NE expEscalar { code(neEE); }
| expEscalar AND expEscalar { code(andEE); }
| expEscalar OR expEscalar { code(orEE); }
| NOT expEscalar            { $$ = $2; code(notE); }

//VECTOR-VECTOR
| expVectorial GT expVectorial { code(gtVV); }
| expVectorial GE expVectorial { code(geVV); }
| expVectorial LT expVectorial { code(ltVV); }
| expVectorial LE expVectorial { code(leVV); }
| expVectorial EQ expVectorial { code(eqVV); }
| expVectorial NE expVectorial { code(neVV); }
| expVectorial AND expVectorial { code(andVV); }
| expVectorial OR expVectorial { code(orVV); }
| NOT expVectorial             { $$ = $2; code(notV); }
```

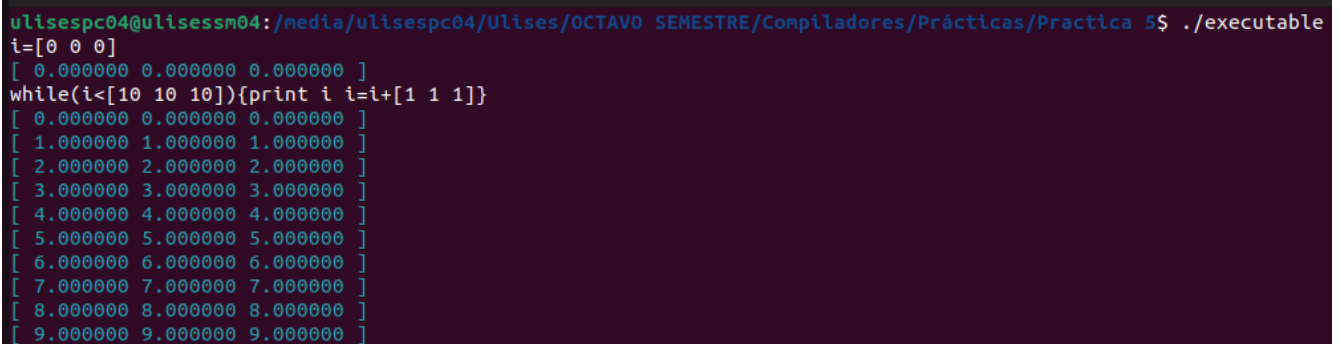
En la máquina nombrada code.c se agregaron las funciones de ifcode y whilecode las cuales nos van a permitir evaluar y ejecutar el ciclo:

```
void whilecode() {
    Datum d;
    Inst *savepc = pc; /* cuerpo de la iteracion */
    execute(savepc+2); /* condicion */
    d = pop();
    while (d.num) {
        execute(*(Inst **)(savepc)); /* cuerpo */
        execute(savepc+2);
        d = pop();
    }

    pc = (*(Inst **)(savepc+1)); /* siguiente proposicion
}

void ifcode(){
    Datum d;
    Inst *savepc = pc; /* parte then */
    execute(savepc+3); /* condicion */
    d = pop();
    if (d.num)
        execute(*(Inst **)(savepc));
    else if (*(Inst **)(savepc+1)) /* parte else? */
        execute(*(Inst **)(savepc+1));
    pc = (*(Inst **)(savepc+2)); /* siguiente proposicion */
}
```

Finalmente se compilan los archivos YACC y C y se procede a ejecutar el programa como se muestra a continuación:



```
ulisespc04@ulisesm04:/media/ulisespc04/Ulises/OCTAVO SEMESTRE/Compiladores/Prácticas/Practica 3$ ./executable
i=[0 0 0]
[ 0.000000 0.000000 0.000000 ]
while(i<[10 10 10]){print i i=i+[1 1 1]}
[ 0.000000 0.000000 0.000000 ]
[ 1.000000 1.000000 1.000000 ]
[ 2.000000 2.000000 2.000000 ]
[ 3.000000 3.000000 3.000000 ]
[ 4.000000 4.000000 4.000000 ]
[ 5.000000 5.000000 5.000000 ]
[ 6.000000 6.000000 6.000000 ]
[ 7.000000 7.000000 7.000000 ]
[ 8.000000 8.000000 8.000000 ]
[ 9.000000 9.000000 9.000000 ]
```

### Conclusiones:

En conclusión esta práctica nos permite ver como es que se puede ejecutar un ciclo al ingresarlo a la terminal, también nos permitió ver la construcciones y como actúa cada una de las funciones en el mapa de memoria, también fue un tanto difícil entender como es que se realiza la comparación y se llego a que todos los elementos ingresados deben de ser del mismo tipo.