

UNIVERSIDAD NACIONAL
AUTÓNOMA DE MÉXICO

FACULTAD DE CIENCIAS

Compiladores Proyecto Final
Manual de uso e instalación

Profesor:

Adrián Ulises Mercado
Martínez

Ayudantes:

Karla Adriana Esquivel
Guzmán
Carlos Gerardo Acosta
Hernández

1. Proyecto Final

1. Requisitos

Se utilizó la versión 3.3.2 de bison.

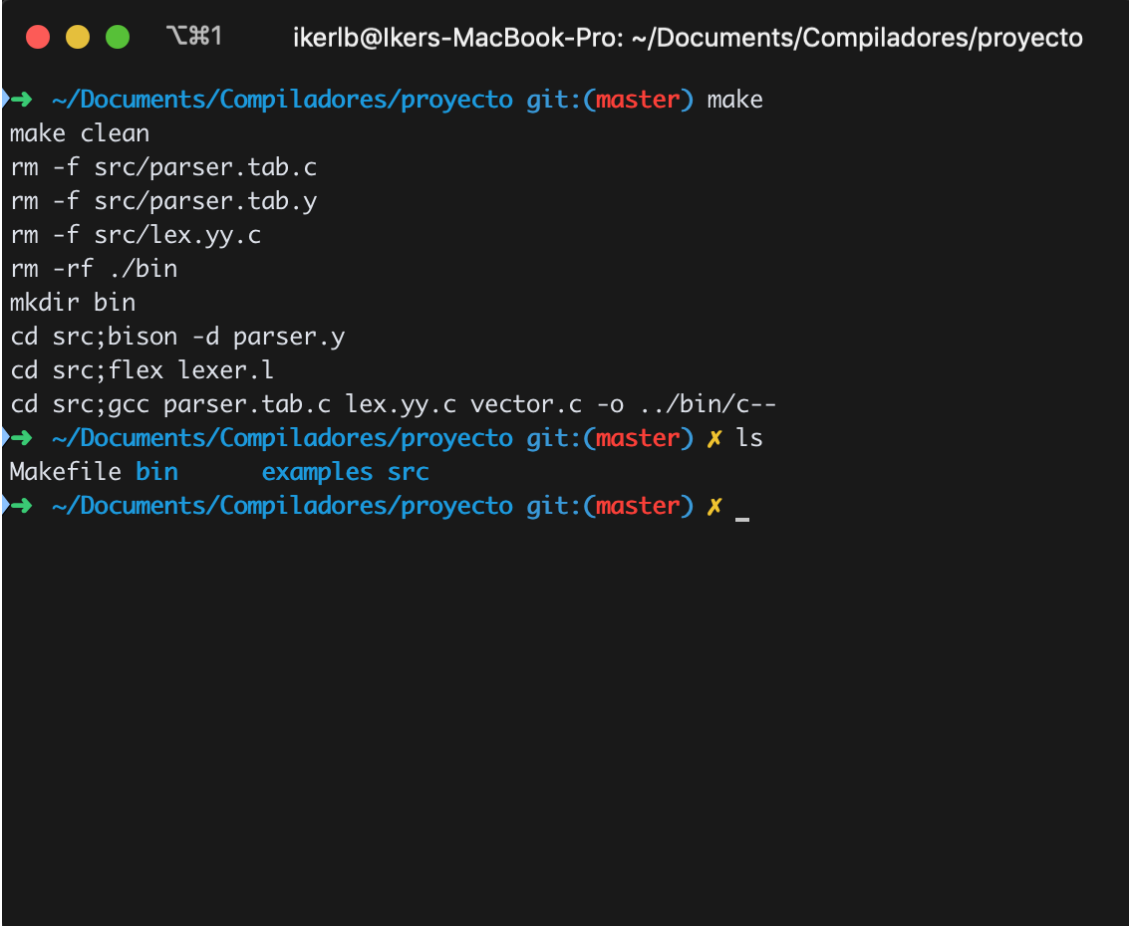
El código fuente se encuentra en:

https://github.com/Ikerlb/compiladores_proyecto

2. Compilación

Para ser compilado el proyecto deberemos de movernos a nuestra carpeta raíz (La que contiene el archivo Makefile). Basta ejecutar el siguiente comando:

```
$ make
```



```
ikerlb@Ikers-MacBook-Pro: ~/Documents/Compiladores/proyecto
➔ ~/Documents/Compiladores/proyecto git:(master) make
make clean
rm -f src/parser.tab.c
rm -f src/parser.tab.y
rm -f src/lex.yy.c
rm -rf ./bin
mkdir bin
cd src;bison -d parser.y
cd src;flex lexer.l
cd src;gcc parser.tab.c lex.yy.c vector.c -o ../bin/c--
➔ ~/Documents/Compiladores/proyecto git:(master) ✖ ls
Makefile bin      examples src
➔ ~/Documents/Compiladores/proyecto git:(master) ✖ _
```

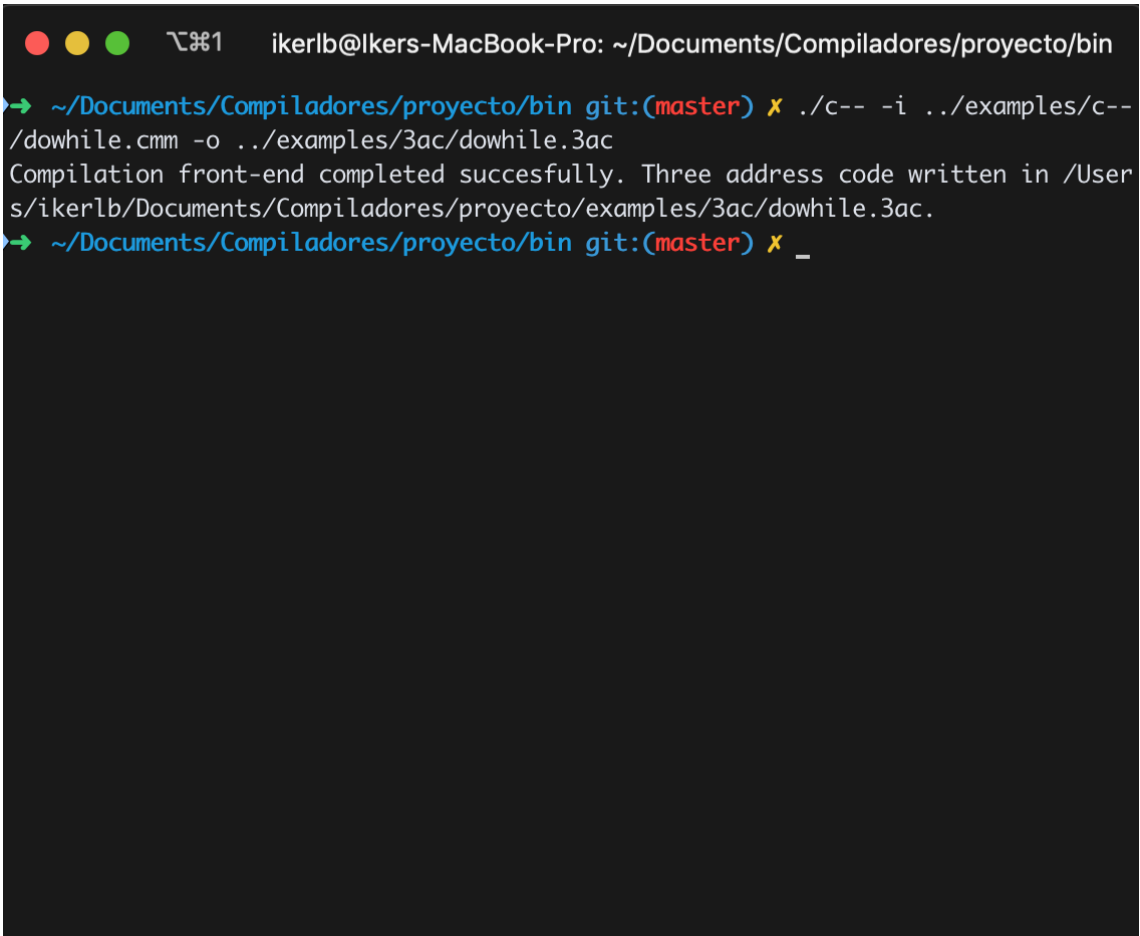
Figura 1: Corriendo make para compilar el proyecto correctamente.

Este comando ejecuta archivo Makefile, que a su vez corre una serie de comandos con el objetivo de compilar correctamente nuestro proyecto.

Una vez creada la carpeta bin (y a su vez el ejecutable c-), procedemos a correrlo de la siguiente forma:

```
$ ./c-- -i programa.cmm -o cod3dir.3ac
```

Donde las banderas -i y -o indican que el siguiente argumento contienen los archivos de entrada y salida respectivamente.



```
ikerlb@Ikers-MacBook-Pro: ~/Documents/Compiladores/proyecto/bin
➔ ~/Documents/Compiladores/proyecto/bin git:(master) ✕ ./c-- -i ../examples/c--/dowhile.cmm -o ../examples/3ac/dowhile.3ac
Compilation front-end completed succesfully. Three address code written in /Users/ikerlb/Documents/Compiladores/proyecto/examples/3ac/dowhile.3ac.
➔ ~/Documents/Compiladores/proyecto/bin git:(master) ✕ _
```

Figura 2: El programa regresa el código de tres direcciones al archivo de salida pasado como argumento.

En este caso, el ejecutable `c-` es nuestro compilador y, en la figura 2 podemos observar que se pasa el archivo `dowhile.cmm`, cuyo contenido es el siguiente.

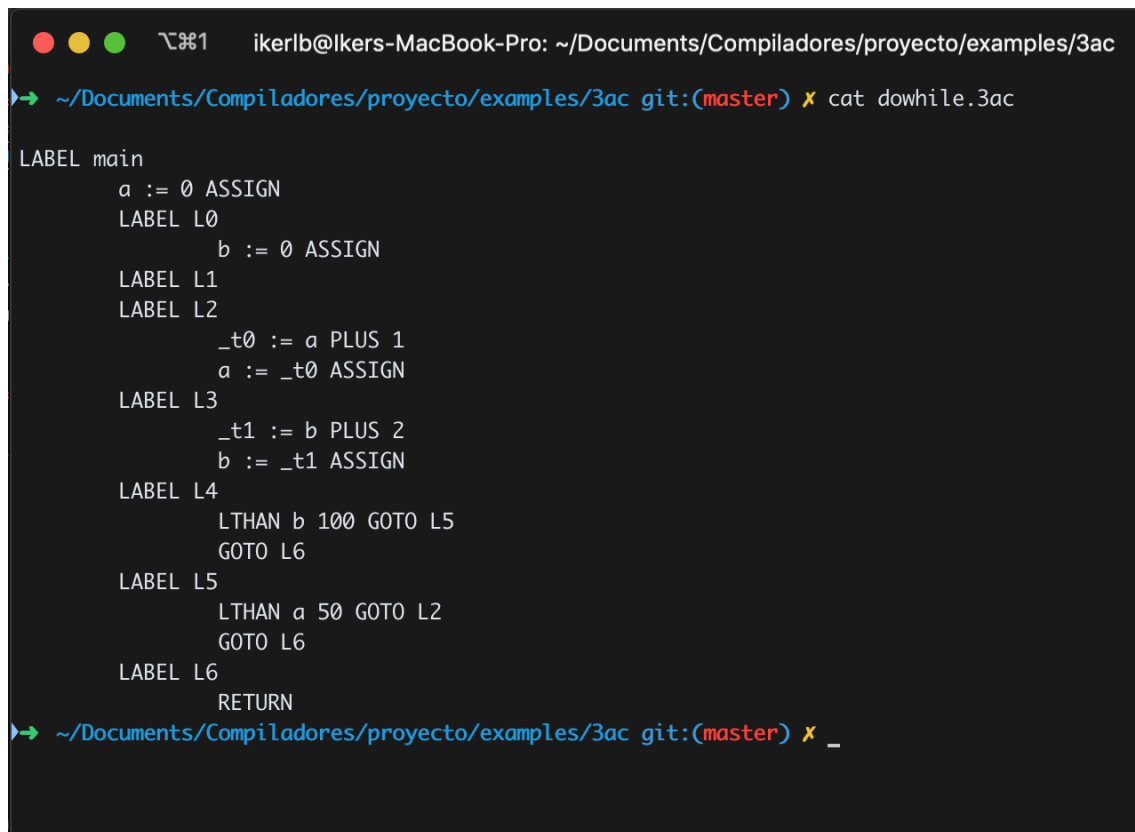
```
vim dowhile.cmm
```

```
_func vacio main(){  
    entero a,b;  
    a=0;  
    b=0;  
    haz {  
        a=a+1;  
        b=b+2;  
    } mientras(b<100 y a<50);  
    regresa;  
}
```

```
"dowhile.cmm" [noeol] 10L, 107C
```

Figura 3: Contenido de archivo dowhile.cmm

Que a su vez genera el siguiente codigo de tres direcciones:



```
ikerlb@Ikers-MacBook-Pro: ~/Documents/Compiladores/proyecto/examples/3ac
➔ ~/Documents/Compiladores/proyecto/examples/3ac git:(master) ✗ cat dowhile.3ac

LABEL main
  a := 0 ASSIGN
  LABEL L0
    b := 0 ASSIGN
  LABEL L1
  LABEL L2
    _t0 := a PLUS 1
    a := _t0 ASSIGN
  LABEL L3
    _t1 := b PLUS 2
    b := _t1 ASSIGN
  LABEL L4
    LTHAN b 100 GOTO L5
    GOTO L6
  LABEL L5
    LTHAN a 50 GOTO L2
    GOTO L6
  LABEL L6
    RETURN
➔ ~/Documents/Compiladores/proyecto/examples/3ac git:(master) ✗ _
```

Figura 4: Contenido de archivo dowhile.cmm

3. Uso

Ejemplos más concretos pueden ser encontrados en el repositorio bajo el directorio ejemplos/c-.

Tipos

Nuestro lenguaje tiene los siguientes tipos:

- a) vacío - corresponde al tipo void
- b) entero - corresponde al tipo int
- c) flotante - corresponde al tipo float
- d) doble - corresponde al tipo double
- e) registro - corresponde a la construcción/tipo struct

Los tipos corresponden a funciones como a variables. Las variables tuvieron que haber sido declaradas antes de usarse. Una variable no puede ser de tipo vacío, únicamente las funciones.

Ejemplo de declaraciones:

- a) entero a,b;
- b) flotante a,b,c;
- c) registro {doble real;doble imaginario;} complejo;

Funciones

Se pueden declarar cualquier cantidad de funciones con identificadores diferentes. Pero es necesario que se hayan declarado y definido antes de hacer llamadas a ellas.

Es además necesario que la función main esté presente en el programa y además sea la última función declarada.

Ejemplo de declaraciones de funciones:

- a) func entero main(){...}
- b) func void setReal(registro {doble real;doble imaginario;} complejo,doble real){...}
- c) func flotante a(){..}

Arreglos

Nuestro lenguaje permite el uso de arreglos y de arreglos de arreglos. Su tamaño debe de ser definido en su declaración. Nuestro lenguaje NO PERMITE el paso de arreglos como parametros

Ejemplo de declaraciones de funciones:

- a) doble d[10][10];
- b) flotante f[100][10][10];

Secuencias

Nuestro lenguaje hace uso de las siguientes estructuras de control:

- a) operacion de asignación =
- b) llamada a funciones (no asignada) para permitir el uso de funciones de tipo void.
- c) while – correspondiente a while
- d) haz-mientras – correspondiente a do-while
- e) para – correspondiente a for
- f) si – correspondiente a if
- g) si-sino correspondiente a if-else

Operaciones booleanas

Las operaciones booleanas primitivas en nuestro lenguaje son las siguientes:

- a) y – correspondiente
- b) o – correspondiente —
- c) no – correspondiente !
- d) operaciones relacionales entre expresiones =, !=, <, >, <=, >=

Expresiones

Nuestro lenguaje considera las siguientes construcciones como expresiones:

- a) identificador
- b) elemento de un arreglo (a[i][j])
- c) número constante de tipo numerico
- d) llamada a funcion no vacia
- e) Campo de un tipo registro. (id.id)
- f) operaciones binarias de expresiones +, -, *, /, %, &
- g) operaciones unarias de expresiones -, +, ~, !

Otras cosas a considerar

Al no tener tiempo suficiente para su implementación, nuestro lenguaje omite las siguientes características:

- a) uso de cadenas y caracteres
- b) las estructuras de control switch y break
- c) paso de arreglos como argumentos
- d) generación de codigo mips