Universidad de San Carlos de Guatemala
Facultad de Ingeniería
Escuela de Ciencias y Sistemas
Organización de Lenguajes y Compiladores 2
Segundo Semestre 2009

Segundo Proyecto

Introducción

Los lenguajes de programación día con día siguen evolucionando, todos los lenguajes de alto nivel poseen ahora interfaces de comunicación entre cada uno de ellos, para su comunicación, y creando funcionalidades similares para cada uno de ellos, con lo que se trata de homogenizar los lenguajes.

Objetivos generales

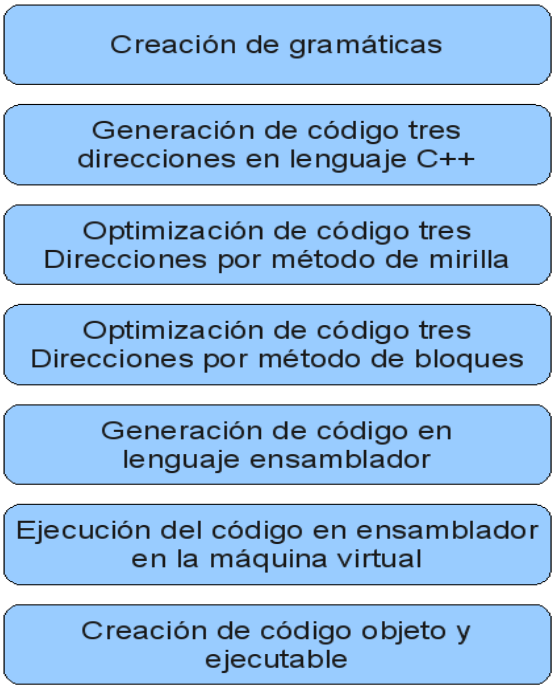
Aplicar los conocimientos aprendidos en clase y laboratorio sobre el proceso y fases de compilación de un programa y comprender la forma en la que funciona internamente.

Objetivos específicos

- Que el estudiante sea capaz de generar gramáticas capaces de reconocer lenguajes de alto nivel.
- Que el estudiante sea capaz de utilizar herramientas de diseño y construcción de gramáticas.
- Que el estudiante sea capaz de reconocer las fases de un compilador.
- Que el estudiante sea capaz de entender el funcionamiento de la pila de memoria interna que se utiliza al ejecutar un programa.

Descripción General

El segundo proyecto del curso de Organización de lenguajes y compiladores 2 consiste en diseñar una aplicación capaz de realizar todas las fases de compilación sobre un lenguaje específico y simular su ejecución mediante una maquina virtual.



Para esto, se realizará de la siguiente forma:

- Se construirá una aplicación tipo IDE, el cual tendrá las siguientes funcionalidades:
    - Creación de proyectos: al momento de generar un nuevo proyecto, la aplicación deberá generar un nuevo proyecto en el cual podrán añadir nuevos archivos, teniendo únicamente un archivo como clase principal.
    - Abrir proyectos: la aplicación deberá ser capaz de leer proyectos creados anteriormente con total transparencia.
    - La función principal del IDE será la de editor de texto, lo cual se explicará más adelante
- Para la parte de compilación, se tendrá un compilador en línea el cual tendrá las siguientes funcionalidades:
    - Compilar los archivos y generar código intermedio.
    - Optimizar el código intermedio.
    - Generar el código objeto en lenguaje ensamblador.
    - Generar el ejecutable de dicho programa.
    - Simulación del proceso del programa compilado.

  Todos los lenguajes utilizarán el paradigma de programación orientada a objetos, utilizando las características de herencia, polimorfismo y abstracción.

  Además de esto, los lenguajes deben trabajar e interactuar entre sí, utilizando una clase principal.

Interfaz Grafica

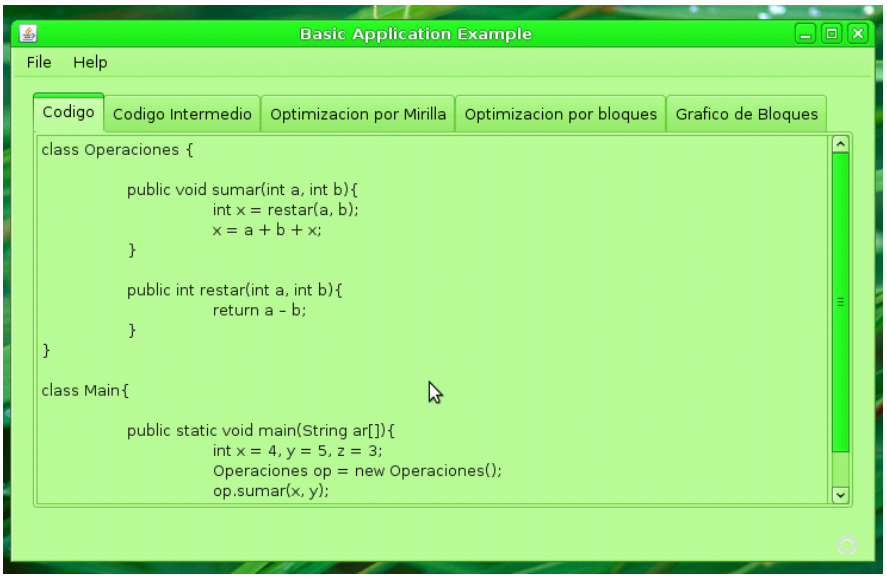
IDE: Mostrará el código fuente de las clases que se quieran compilar. Recuerde que las entradas pueden ser varios archivos en los distintos lenguajes que se implementaran, definidos en la siguiente sección.

Debe tener las siguientes características

- Color de palabras reservadas
    - Tipo de datos : verde
    - Palabras reservadas : azul
- Color de errores
    - Los errores deben mostrarse gráficamente, subrayando el error con rojo, con la opción de que al darle un clic encima de la palabra, este muestre un menú indicando cual sería la solución apropiada o la palabra correcta que se debe incluir.
    - Esto debe ser para los errores léxicos y sintácticos.
- Métodos y atributos
    - Debe existir una sección en la cual se pueda visualizar los atributos y los métodos del archivo activo.
    - Se debe indicar en qué método se encuentra el cursor, esto de forma gráfica.

Además, el IDE debe trabajar con tabs para mayor comodidad de visualización de los archivos.

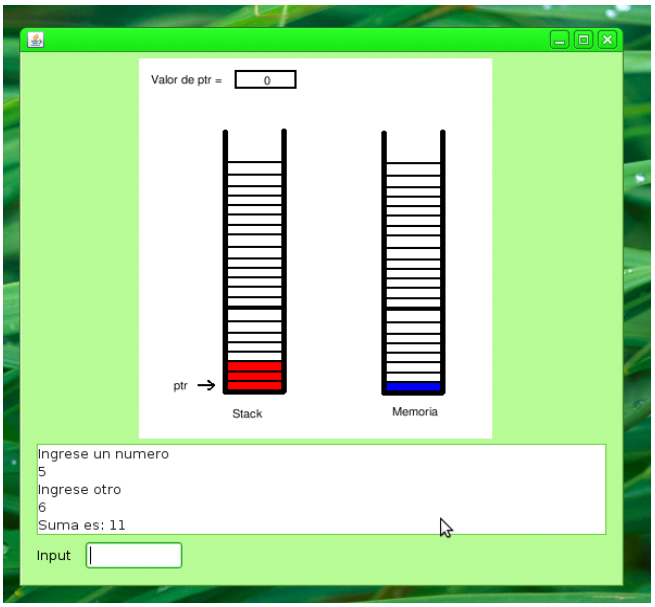
La presentación de los archivos, así como los colores y el menú de errores quedan a discreción del estudiante y se tomará muy en cuenta la creatividad de cada uno.



Compilador en Línea:

Se creara una aplicación web, éste tendrá la funcionalidad de realizar las fases de compilación para el proyecto generado en el IDE, teniendo las siguientes características

- Carga del proyecto:
    - En la página principal se tendrá la opción de cargar el proyecto al servidor, esto puede ser de dos formas
        - Cargando archivo por archivo
        - Cargando un archivo comprimido en formato Zip
- Compilación del proyecto:
    - La aplicación generará el código de tres direcciones para el proyecto sin errores y lo mostrará en pantalla, indicando si existió algún error en compilación, en este caso solamente errores semánticos
- Optimización:
    - La aplicación deberá poseer las siguientes opciones de optimización:
        - Optimización por mirilla:
            - Se debe llevar un log detallado de las sentencias que se están optimizando.
            - Se debe de mostrar el resultado de la optimización, esto teniendo en una sola ventana el código normal y el código optimizado, además del log.
        - Optimización por bloques:
            - Se debe mostrar un grafo, el cual contendrá los nodos que contendrán los bloques de código. Al seleccionar un nodo debe mostrar:
                - Código líder
                - Código arista
                - Código dentro del bloque
- Generación de Código Objeto:
    - La aplicación debe generar el código ensamblador del código intermedio antes generado, el cual debe mostrarse en pantalla definiendo colores para las palabras reservadas.
- Maquina virtual:
    - La aplicación dará la opción de simular la ejecución del programa ensamblador ya sea de un proyecto cargado y llevado por las diferentes fases y el de cargar un archivo .asm
    - Consistirá en mostrar gráficamente la pila y la memoria al momento de ejecutar, así como los registros del procesador.



- Archivos generados
  - Código tres direcciones en código C++ (.cpp)
  - Código tres direcciones optimizado por mirilla (.cpp)
  - Código tres direcciones optimizado por bloques (.cpp)
  - Código en lenguaje ensamblador ASM (.asm)
  - Código objeto (.obj)
  - Programa ejecutable (.exe)
  - Dibujo del grafo generado para la optimización de bloques (.jpg)
  - Reporte de errores semánticos
  - Para cada uno de estos archivos, la aplicación deberá tener una opción de descarga

Lenguajes

Descripción:

- Como se indico anteriormente, solo en la clase principal de c++ pueden crearse instancias de las otras clases, por ejemplo; la clase principal de C++ puede crear una instancia de una clase de java o de visual basic o ya sea de python.
- Solamente en la clase principal de C++ se utilizará la sentencia #include para añadir las clases, para los demás lenguajes será de manera implícita.
- Todas las clases deben poseer un constructor.
- Las clases del mismo lenguaje pueden crear instancias, por ejemplo; las clases de java solo pueden crear instancias de otras clases java.

- **C++**
  - **Especificaciones**
    - Para crear un archivo C++ valido, deberá crearse un archivo .h el cual llevara las definiciones de los procedimientos y los atributos
    - Y un archivo .cpp el cual incluirá todo el código de los procedimientos, para este proyecto solo se podrán crear procedimientos definidos dentro de la clase
    - Para cada proyecto generado en la aplicación IDE, solo una clase generada en C++ podrá ser la clase principal, en el cual deberá crearse una función Main con la cual se ejecutara todo el código
    - Además, solo en la clase principal se podrán hacer instancias de las demás clases creadas en un lenguaje diferente, o sea que la clase principal será la única que podrá acceder a métodos de otras clases que no sean de su mismo lenguaje.

  - **Librerías**

    #include "nombredeClase"
  - **Definición de Clases**
    - Las clases podrán ser de tres tipos
      - Normales

        Class MiClase {
        //sentencias
        }
      - Heredadas

        Class OtraClase : public MiClase {
        //sentencias
        }

      - Abstractas: definiendo los métodos como abstractos

```
class Abstracta{
public:
        virtual int metodo() = 0;
};
```

- **Definición de variables**: Tipos de datos int, float, char, double, long, otras clases definidas por el usuario, para efectos del proyecto también se utilizar el tipo string dentro de los tipos de variables

```
int x = 0;
int y = CONSTANTE;
char caracteres[20]
caracteres[6] = 'c';
String cadenas[3]
cadenas[0] = "esta es la primera cadena del arreglo";
cadenas[2] = "esta es la tercera cadena del arreglo";
float f = 14.5f;
double d = 14.5d;
```

- **Definición de procedimientos**

```
int nombreClase::procedimiento1() {
        //sentencias
        return 0;
}

Float nombreClase::procedimiento2(string cadena, int numero)
{
        //sentencias
        return 0.0;
}
```

Nota: la visibilidad de los procedimiento será definido en el archivo .h, además todos los procedimientos pueden retornar de cualquier tipo de dato.

- **Llamadas a procedimientos**

```
inicializa()
suma(x,y)
suma(5,y)
suma(6,x[8][3])
suma(7,getX())
suma(resta(6,y),7)
x = suma(x,y)
y = suma(x,y) + resta(y,x)
```

- **Estructuras de control**

  - **Condiciones if**

```
if(x > y){
    //sentencias
}

if( x <= 5){
    //sentencias
} else if (x > 5){
    //sentencias
}

if(mat[x][y] == funcionBooleana()){
    //sentencias
} else {
    //sentencias
}
```

    Nota: cuando se utiliza únicamente una sentencia, no se utiliza las llaves, por ejemplo

```
If (x>n)
    //sentencia
```

  - **Ciclos FOR con condición simple/compuesta**

```
for (int x = 0; x<y; x++) {
    //código
}
for (int x = y; x<z; x+=3) {
    //código
}
for (int x = z; x>=6 && y<3; x) {
    //código
}
for (x = x+3; x<16; ++x) {
      y = x*15;
}
```

  - **Ciclos WHILE con condiciones simples y/o complejas**

```
while (x>y) {
    //código
}
```

```
while (x<=4) {
    //código
}
while (x[6][5]>y[3]) {
    //código
}
while (x>y && y[5]==4) {
    //código
}
while (x>y || y[5]==4){
    //código
}
```

- **Ciclos DO-WHILE con condición simple y/o complejas**

```
do{
    //código
} while (x>y);
do{
    //código
} while (x<=4);
do{
    //código
} while (x[6][5]>y[3]);
```

- **Sentencias SWITCH-CASE**

```
switch (x) {
     case  0  :  ... break;
     case  1  :  ... break;
     case  2  :  ...
     case  3   :  ...
     case 4   :  ... return ...
     default   :  ...
}
switch (y[7]) {
     case  0   :  ... break;
     case  1   :  ... break;
}
```

- **Visibilidad**: es definida para cualquier atributo y procedimiento dentro del archivo .h
  - Public: puede ser visto desde cualquier lado utilizando una instancia del mismo.
  - Private: solo puede ser visto desde la misma clase.
  - Protected: significa que pueden ser visto desde cualquier instancia solamente cuando se hereda de esta clase.

- Static: se utiliza para indicar que el atributo o procedimiento posee la misma posición de memoria en toda la ejecución del programa.
- Final: indica que una variable es constante

- **Java**
  - **Especificaciones**

    Lenguaje multiplataforma interpretado por una máquina virtual, desarrollado por Sun Microsystem, implementa el paradigma de programación orientado a objetos.

  - **Definición de Clases**

```
public class MiClase{
        //código
}

public class MiOtraClase extends MiClase{
        //código
}

Abstract class miClaseAbstracta {
        //código
}
```

  - **Definición de variables:** tipos de datos int, float, char, double, long, String, otras clases definidas por el usuario

```
int x = 0;
int Z;
int CONSTANTE = 65;
int y = CONSTANTE;
String nombre = “Compi2”;
String dato = nombre + “ Seccion A”;
char[] caracteres = new char[70];
caracteres[6] = ‘c’;
String cadenas = new String();
cadenas[0] = “esta es la primera cadena del arreglo”;
cadenas[2] = “esta es la tercera cadena del arreglo”;
float f = 14.5f;
double d = 14.5d;
double d2 = 14.5;
```

  - **Definición de procedimientos**

```
public void procedimiento1(){
        //código
```

```
}

public static int sumando(int a, int b){
        //código
}

public Objeto getObjeto(){
        Objeto obj;
        //código
        return obj;
}
```

- **Llamadas a procedimientos**

  Las mismas de C++

- **Estructuras de control**

  Las mismas de C++.

- **Visibilidad**

  Las mismas de C++

- **Visual Basic**
  - **Especificaciones**

    Lenguaje compilado en sus versiones anteriores (versión 6 y posteriores), actualmente en la versión 2008 de la plataforma .NET, implementa el paradigma de programación orientada a objetos. También con la característica de ser multiplataforma (para M$ Windows y MacOS).

  - **Definición de Clases**

```
Class VBClase Begin
        //código
End

public class BasicClase : VBClase Begin
        //código
End Class
```

  - **Definición de variables**

```
x As Integer;
y As Float;
str As String, numero As Integer;
Public num As Double;
Private Static caracter As Char;
Dim arreglo(5) as Integer
VBObj As New Objeto;
VBEjemplo As New Calculadora(param1, param2);
```

```
Dim vec(10) As Integer
vec(1) = 4;
vec(3) = 6;

Redim vec(20) As Integer
vec(20) = 5;
```

- **Definición de procedimientos**

```
public sub procedimientoVB()
        //código
end sub

function sumandoIVA(cantidad as Double) as Double
        //código
        sumandoIVA = cantidad * 1.12;
end function
```

- **Llamadas a procedimientos**

```
Call procedimiento
Iva = sumandoIVA(123.00)
```

- **Estructuras de control**

  - **Condiciones if**

```
if(a > b and 6 < vec[funcionBooleana()] or a+b*z == 77) then
        //código
end

if (x > y and 5 == funcion() or var != var2)  then
        //código
else if( a <  5) then
        //mas Código
end
```

  - **Ciclos FOR con condición simple/compuesta**

```
for n = 0 to 99 do
        //código
next n

for y = 4 to 100 do
        //código
next y
```

- **Ciclos WHILE con condiciones simples y/o complejas**

```
do while (x>y and y[5]==4 or z++<1)
     y = x*15;
loop

do while (x>y && y[5]==4)
          //código
loop
```

- **Ciclos DO-WHILE con condición simple y/o complejas**

```
repeat
          //código
until (x>y and y[5]==4);

repeat
          //código
 until (x>y or y[5]==4);

repeat
          //código
until (x>y and y[5]==4 or  z++<1);
```

- **Visibilidad**

  Las mismas que C++

- **Python**
  - **Especificaciones**

    Lenguaje multiplataforma y multiparadigma, es decir, puede programarse estructurado, orientado a objetos y programación funcional, es un lenguaje interpretado administrado por Python Software Fundation.

  - **Definición de Clases**

```
class clasePy:
          //código
class otraClasePy:
          //código
```

  - **Definición de variables**

- Por ser un lenguaje de alto nivel, no necesita declarar el tipo de variable, depende del uso que se tome ese será su tipo por default: string, int, float, double, boolean
- El mètodo constructor de python es

```
__init__(self):
        //cuerpo del mètodo
```

  En donde esta self, pueden ir mas paràmetros si lo requiere.
- El àmbito de las variables y objetos dependen de la tabulacion del còdigo, por ejemplo:

```
def usoUnFor(self):
    for a in range(10,20):
        if a/3 < 5:
            print (a)
        print("dentro del for")
    print("afuera del for)
```

  El for esta dentro del mètodo usoUnFor, luego dentro del for hay un if y este tiene un print, luego en la siguiente linea viene otro print, pero no es parte del if, por la tabulacion es parte del for y por ultimo se ve un ultimo print que esta al nivel de tabulacion del for, es decir, esta fuera del for.

- Los tipos de variables definidos por el usuario se definen de esta forma.

```
obj1 = new Objeto();
obj2 = new Ventana(param1, param2);
```

- Los arreglos se definirán de la siguiente forma:

```
mlist1 = [[7, 12, 23],[22, 31, 9],[4, 17, 31]]

mlist1[1][2] = 99
```

o Definición de procedimientos

```
def sumando(self):
    a + b

def funcioncita(param1, param2):
    return param1 + param2
```

o Estructuras de control

- **Condiciones if**

```
if (num % 2) == 0:
    //código
```

```
elif (num % 2) == 0 and num > 100:
        //otro Código
else
        //mas Código
```

- **Ciclos FOR con condición simple/compuesta**

```
for i in range(1, 5):
    //código
```

- **Ciclos WHILE con condiciones simples y/o complejas**

```
while (x> y) :
        //código
```

Métodos especiales de lectura y escritura

Lectura: Se usaran métodos especiales y bien definidos para la lectura de datos desde la consola

- **Leer enteros:** readInt()
- **Leer decimales:** readFloat()
- **Leer caracteres:** readChar()
- **Leer decimales grandes:** readDouble()
- **Leer enteros grandes:** readLong()
- **Leer cadenas:** readString()
- **Pausar el cursor, solo para c++:** getch()

Escritura: Se usaran métodos especiales para desplegar datos a la consola

- **Imprimir y quedarse en la misma línea:** print()
- **Imprimir y bajar de línea:** println()

Ejemplos

- print(“Hola”)
- print(variable)
- print(var1+ var2)
- println(“esta es el valor esperado”+var8)
- println(“resultado de un método”+metodo())

Operadores aritméticos binarios:

- suma +
- resta -
- multiplicación *
- división /
- residuo %

Operadores booleanos

- mayor >
- menor <

- mayor que >=
- menor que <=
- igual de comparación ==
- distinto !=, <>

Operador unario
- menos -

Operador post y pre incremento
- i++
- ++i

Operador post y pre decremento
- o--
- --o

Nota: Los operadores pre y pos incremento/decremento solo existen en java y c++

Operadores especiales
- +=
- -=
- *=
- /=
- %=

Operadores lógicos
'y' lógico
- AND para visual basic
- &&
- &

'o' lógico
- OR para visual basic
- | |
- |

'no' lógico
- NOT para visual basic
- !

Nota: Los operadores lógicos de java, c++ y python son los mismos

Referencia a objetos this y super, esto solo para java y c++

```
this.variable;
this.getVariable();
this.setVariable();
```

```
super.setVariable();
super();
```

Para python se utiliza **self**

Las referencias que no hayan sido inicializadas en Java y Basic tendrán un valor de **null**, en python tendrán el valor de **none**.

Comentarios

Para comentarios de una sola línea, se aplicaran en los tres lenguajes

```
// Este es un comentario de una sola línea
```

Para comentarios multilínea se tendrá

```
/*
Este
es
un
comentario multilínea */
```

Generación de código intermedio

Para el codigo intermedio, se utilizará la nomeclatura utilizada en clase, el manejos de memoria, etiquetas y metodos auxiliares para la creacion de codigo intermedio queda a discreción del estudiante, debe realizarse en un archivo .cpp.

Optimización de código

La optimización de código se realizará sobre el codigo intermedio generado en el paso enterior, un ejemplo basico de eso se muestra a continuación:

```
b = a;
c = b;
d = c;
```

Optimizado quedaría:

```
d = a;
```

El número de asignaciones se reduce y por lo tanto el programa se ejecutará más rápido.

El programa debe ser capaz de optimizar el código intermedio para un funcionamiento más eficiente. Los metodos de optimización para el proyecto son:

Mirilla: Se tomará como entrada, el código tres direcciones generado en la sección anterior. El usuario debe poder configurar el tamaño de la ventana de la mirilla y debe poder ver como se está optimizando el código intermedio. La ventana en mirilla se define como la cantidad de líneas que van a ser procesadas al optimizar el código, por ejemplo, si se toma una ventana de 4, serán 4 líneas de código procesadas para optimizar. Al usar mirilla se pueden hacer varias iteraciones, pues en una sola pasada puede que se omitan líneas a optimizar, las siguientes iteraciones seguirán optimizando hasta cuando ya no hayan cambios en el código. Lo que se optimiza con esta técnica es:

- Cargas de almacenamiento redundante

- Código inalcanzable
- Código muerto
- Simplificación algebraica
- Reducción intensidad

Bloques: Se deberá tomar como entrada el código optimizado por el método de mirilla. Con este método deberán generar los grafos correspondientes para indicar los bloques, código líder, código aristas que indican saltos y el código que forma cada bloque. Este método debe mostrar:

- Listado de bloques básicos y su contenido
- Elemento líder de cada bloque
- Dibujo del grafo antes de optimizar.
- Dibujo del grafo después de la optimización.
- Igual que en mirilla se debe poder ver como se está optimizando el código intermedio generado.

Al terminar cada optimización se mostrará el código optimizado y el código original.

Lenguaje ensamblador generado

Luego de optimizar el código en tres direcciones se deberá a traducir este a código en lenguaje ensamblador para procesadores de la familia x86, estos archivos tienen extensión **.asm**.

Programas objeto y ejecutables

Se deberá usar las herramientas **tasm.exe** y **tlink.exe** incluidas junto al compilador Borland C++. Estas herramientas permiten crear el programa objeto y el programa ejecutable respectivamente. La aplicación deberá hacerlo automáticamente y por ultimo indicar la opción de descarga para ejecutarlo de manera local.

Maquina virtual

Una maquina virtual es un software que es capaz de interpretar y ejecutar instrucciones una por una, en este caso, código tres direcciones y código ensamblador el cual será generado por la aplicación. Adicionalmente se mostrará una consola en la cual se ejecutara el programa. Simultáneamente se mostrará la ejecución de la pila, la memoria y sus apuntadores gráficamente para la depuración del programa.

El código tres direcciones debe poderse compilar y ejecutar con un compilador C++ y su ejecución debe reflejar la lógica del código fuente.

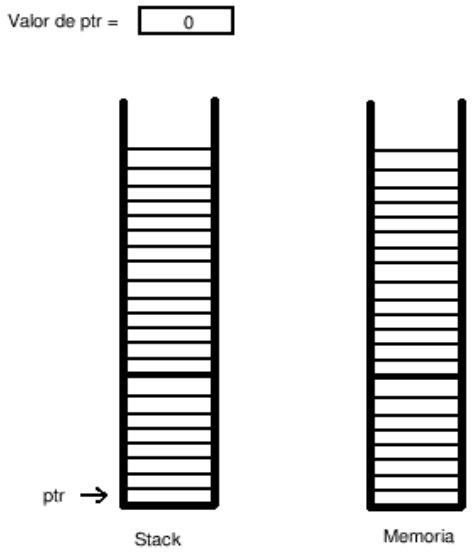
El código en lenguaje ensamblador debe poderse ensamblar para poder obtener el programa objeto, luego el programa ejecutable y su ejecución debe reflejar la lógica del código fuente.

Este es un ejemplo muy básico, ya que los archivos de entrada siempre tendrán un archivo de cada lenguaje y una clase principal en c++, más adelante se encuentra un ejemplo completo y se les estará enviando más archivos de entrada para que puedan probar sus gramáticas.

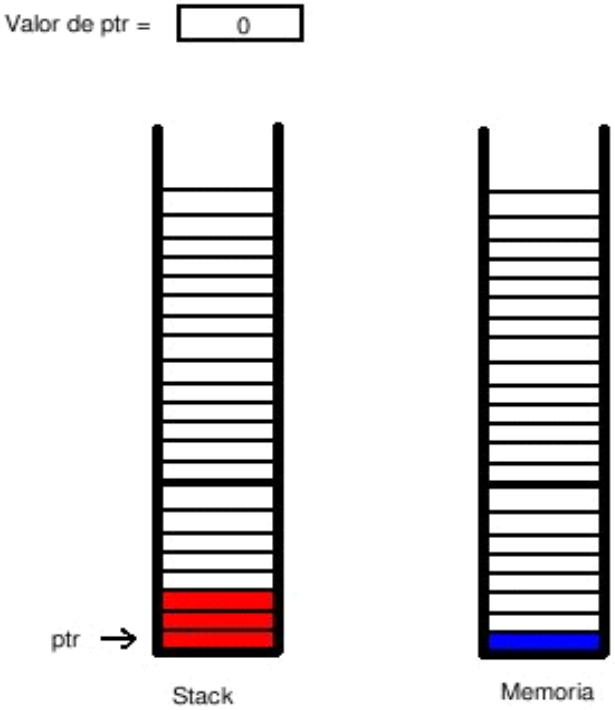
Código de ejemplo:

```
class Operaciones {

	public void sumar(int a, int b){
		int x = restar(a, b);
		x = a + b + x;
	}

	public int restar(int a, int b){
		return a – b;
	}
}

class Main{

	public static void main(String ar[]){
		int x = 4, y = 5, z = 3;
		Operaciones op = new Operaciones();
		op.sumar(x, y);
	}
}
```

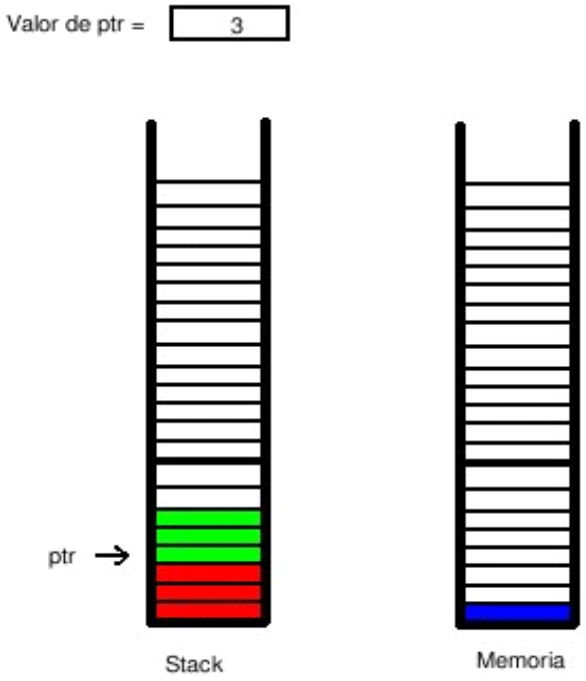
Antes de comenzar el programa la pila y la memoria deben estar vacías



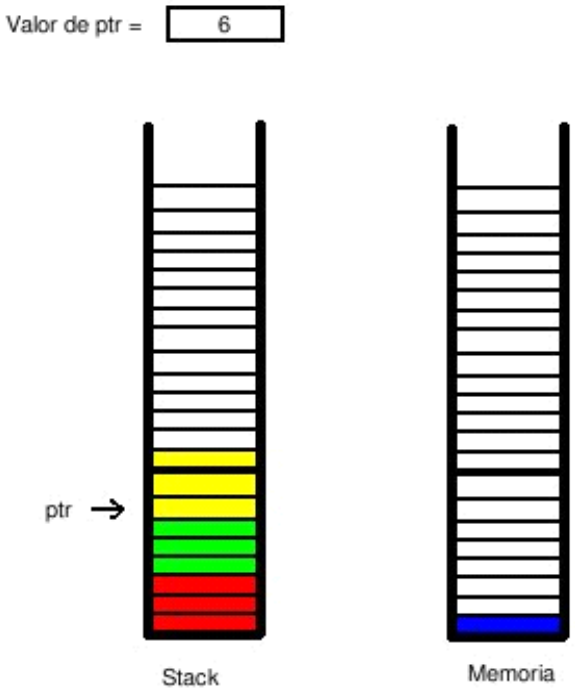
Al iniciar el programa se empezará a llenar la pila o la memoria según las instrucciones del código. En este caso estamos en el ámbito del método main y lo dibujamos con rojo



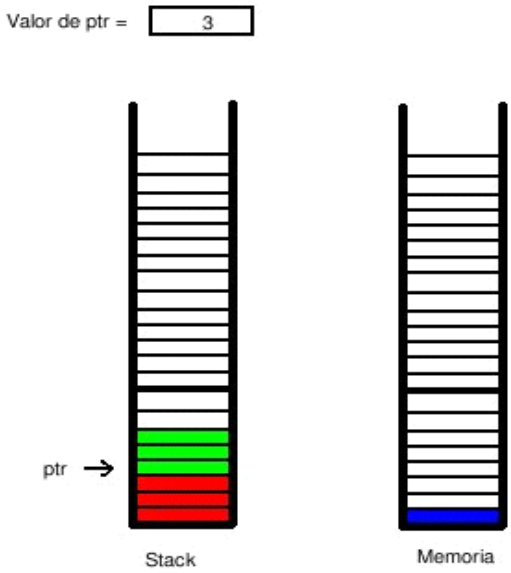
Se llama al siguiente método y se llenara la pila con otro color mostrando los ámbitos de los métodos con colores, en este caso llamando al método sumar, en este caso con color verde.



Moviendo la variable ptr conforme se esté ejecutando el programa. Ahora entra en el ámbito de restar que dibujamos con amarillo.



Cuando termina la ejecución de un método, se regresa el ptr al ámbito anterior, en este caso al método sumar, que esta coloreado de verde.



El método de mostrar la información gráfica de la pila y la memoria estará a discreción del estudiante. Además de la pila (stack) y la memoria (heap) se deberán mostrar los registros de deberán representar de forma gráfica los registros generales y específicos del procesador, el registro de banderas que sean usados en el código ensamblador generado.

Manuales

Manual técnico

Se hará un análisis y diseño del proyecto, se deberá entregar:

- Diagrama de clases
- Diagrama de secuencias
- Casos de uso y sus especificaciones

Se separará en módulos y se explicará el funcionamiento de cada uno, sus entradas, salidas y los procesos que tiene cada uno de ellos.

Manual de usuario

El manual de usuario mostrara como es que funciona el programa de forma amigable e intuitiva. Deberá tener todas sus secciones bien detalladas de manera que el usuario final pueda entender y usar la aplicación.

Tecnologías a utilizar

- Para el desarrollo de la aplicación se usara el lenguaje Java 1.6.0_16
- Se usara cualquier IDE que use el compilador de Java, ya sea NetBeans 6.5, Eclipse, Block de notas, etc.
- Para el análisis léxico deberán usar Jlex
- Para el análisis sintáctico deberán usar Java Cup
- Sistema operativo para el desarrollo: Windows XP/Vista

Restricciones del proyecto

- El proyecto se realizará de forma individual
- La generación de código tres direcciones y código ensamblador debe ser creado EXCLUSIVAMENTE por el compilador implementado en el proyecto, no se aceptarán herramientas que generen dicho código.
- El proyecto debe ser entregado en un CD con todas las fuentes, manuales y archivos de ejemplo con sus respectivas salidas.
- Se valorará la calidad de la información proporcionada por el compilador cuando este produzca errores.
- Copias de proyecto, gramáticas tendrán una nota de cero puntos incluyéndose sus respectivas sanciones de acuerdo al reglamento de la escuela de Ciencias y Sistemas.
- La calificación durará como máximo 30 minutos en un horario posteriormente establecido.
- La fecha de entrega del proyecto será el día 23 de noviembre, sin opción a prorroga y se calificará ese mismo día.

Entregas

Las entregas serán por fases las cuales estarán distribuidas de la siguiente manera:

Entrega #	Contenido	Fecha
1	Análisis y diseño de la aplicación, diagrama de clases, casos de uso y de secuencias (manual técnico).	19/10/2009
2	Gramáticas de los tres lenguajes	26/10/2009
3	Acciones para la generación de código tres direcciones en los tres lenguajes	02/11/2009
4	Optimización por mirilla y bloques	09/11/2009
5	Generación de código en lenguaje ensamblador	16/11/2009
6	Implementación de máquina virtual	21/11/2009

La entrega es antes de media noche de las fechas indicadas, correos que sean recibidos después, no serán aceptados.

Detalles de cada fase:

- La primera fase es un PDF con la documentación del proyecto.
- La segunda son los archivos .lex y .cup con las gramáticas de los tres lenguajes.
- La tercera fase son los archivos .lex y .cup con las acciones de código tres direcciones.
- La cuarta fase son el modulo de optimización por mirilla y bloques que tiene por entrada código tres direcciones generado (archivos **.cpp**).
- La quinta fase son los archivos .lex y .cup con la gramática y acciones para generar código ensamblador.
- La 6ta fase será el módulo de la máquina virtual, ejecutando código ensamblador y tiene por entrada código ensamblador generado (archivos **.asm**) e integrado con las herramientas para generación de código objeto y código ejecutable.

Ejemplo de Código

Calculadora Simple

- Clase Principal C++
    - Archivo .h

```
#include "javaClass"
#include "pythonClass"
#include "visualClass"

class Calculadora {
        private:
                bool siguiente;
                float a;
                float b;
                float resultado;
                int opcion;
                javaClass Resta;
                pythonClass Multi;
                visualClass divRaiz;

        public:
                float suma(int a, int b);
                void exit();

};
```

- Archivo .cpp

```
float Calculadora::suma() {
    println("============ Suma de 2 Números ============");
    println("Introduce un numero:");
    this.a = readFloat();
    println("Introduce otro numero:");
    this.b = readFloat();
    this.resultado = this.a + this.b;

    println("============ Resultados ============");
    println("El resultado es:" + this.resultado);
    getch();
```

```
	return this.resultado;
}

void Calculadora::exit()
{
	println("Programa de prueba");
	getch();
}

int main()
{
	do
	{
		println("============ Calculadora ============");
		println("Escoja una opcion:");
		println("1.- SUMA");
		println("2.- RESTA");
		println("3.- MULTIPLICACION");
		println("4.- DIVISION");
		println("5.- SALIR");
		println("================================");
		this.opcion = readInt();

		switch (this.opcion)
		{
			case 1:
				resultado = Suma ();
			break;

			case 2:
				resultado = Resta.Resta ();
				getch();
			break;

			case 3:
				resultado = Multi.Multiplicacion ();
				getch();
			break;

			case 4:
				resultado = divRaiz.Division ();
				getch();
```

```
                        break;

                        case 5:
                                exit();
                        break;
                        default:
                                println("LA OPCION NO ES VALIDA ");
                                println(" SELECCIONAR NUEVAMENTE LA OPCION ");
                                getch();
                                break;
                }

        }while(this.opcion != 5);
        return 0;
}
}
```

- Clase Java

```
class javaClass {

        private float numero1, numero2;
        private float resultado;

        public javaClass() {
                numero1 = 0.0;
                numero2 = 0.0;
                resultado = 0.0;
        }

        public float Resta()
        {
                println("============ Resta de 2 Numeros ============");
                println("Introduce un numero:");
                this.numero1 = readFloat();
                println("Introduce otro numero:");
                this.numero2 = readFloat();
                this.resultado = this.numero1 - this.numero2;

                println("============ Resultados ============");
                println("El resultado es:" + this.resultado);
                return this.resultado;
```

```
        }
    }
```

- Clase python

```
class Multi:
    numero1 = 0.0;
    numero2 = 0.0;
    resultado = 0.0;

    def Multiplicacion():
        println("============ Multiplicacion de 2 Numeros ============")
        println("Introduce un numero:")
        numero1 = readFloat()
        println("Introduce otro numero:")
        numero2 = readFloat()
        resultado = numero1 * numero2


        println("============ Resultados ============")
        println("El resultado es:" + this.resultado)
        return resultado
```

- Clase Visual Basic

```
public class visualClass

    private numero1 as Float
    private numero2 as Float
    private resultado as float

    function dividir() as float
        println("============ division de 2 Numeros ============")
        println("Introduce un numero:")
        numero1 = readFloat()
        println("Introduce otro numero:")
        numero2 = readFloat()
        if (numero2 != 0)
        begin
            resultado = numero1 / numero2
        end
        else
```

```
                begin
                        resultado = -1
                end

                println("============ Resultados ============")
                println("El resultado es:" + this.resultado)
                return resultado
        end function

end Class
```