ST0244 Lenguajes de programación 031 **Universidad EAFIT**
Primer Parcial **DIS**
Juan Francisco Cardona McCormick **29 de Agosto de 2013**

Nombre: _____

Código: _____

Indicación: Sea breve en sus respuestas y justifiquelas concisamente. Excepto las preguntas de selección múltiple.

1. (20 points) **Ambigüedad**

   Dada la siguiente gramática G_1 que define el lenguaje de las proposiciones lógicas:

$$\begin{array}{rcl} P & \rightarrow & P \wedge P \\ & | & P \vee P \\ & | & P \Longrightarrow P \\ & | & P \Longleftrightarrow P \\ & | & \neg P \\ & | & '('P')' \\ & | & 'T' \\ & | & 'F' \\ & | & \text{identifier} \end{array}$$

   Encuentre si las siguientes cadenas generadas por la anterior gramática son ambiguas:

   a) $\neg a \wedge (a \vee b)$
   b) $a \wedge b \wedge c \vee d$

   Recuerde genere las derivaciones correspondientes y los árboles de derivación que muestre que son ambiguas.

2. (20 points) **Calculadora**

   Dada la siguiente expresión:

$$((3S - 2)S + R)/3S$$

   Responda:

   a) ¿Cuál es el valor de la expresión anterior?
   b) ¿Cuál es el valor al final de la evaluación del registro de memoria?
   c) Muestre el árbol abstracto sintáctico de la anterior expresión.

3. (20 points) **EWE**

   Escriba un programa en ewe que lea cuatro números enteros que representa la longitud de las patas de una mesa. Si los cuatro números son iguales, se puede construir un mesa de cuatro patas; pero si solamente tres de ellas son iguales se puede producir una mesa de tres patas; pero si las dos anteriores condiciones no son válidas, no se puede contruir una mesa.

4. (20 points) **Gramática en BNF**

   Describa el lenguaje definido por la siguiente gramática. Y muestre con un ejemplo lo que produce.

```
<S> ::= <P><N>
<N> ::= <D><N> | <L><N> | <D> | <L>
<P> ::= 0<Y>
<D> ::= [ 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 ]
<Y> ::= x
<L> ::= a | b | c | d | e | f
```

5. (20 points) **Historia del lenguajes de programación**
   Responda las siguientes preguntas de los siguientes lenguajes de programación basada en la lectura del capítulo del libro de M. Gabbrielli y S. Martini sobre la historia de los lenguajes de programación.

   a) ¿El lenguaje de programación PASCAL es un descendiente directo de que lenguaje de programación?

   b) ¿Quién es el creador del lenguaje Smalltalk?

   c) ¿Cuál es el significado del acrónimo de COBOL?

   d) ¿Qué compañía creó el lenguaje de programación COBOL?

   e) ¿Qué significa el acrónimo de FORTRAN?

ST0244 Lenguajes de programación **Universidad EAFIT**
Primer Parcial **DIS**
Juan Francisco Cardona McCormick **26 de Agosto de 2013**

Nombre: _____

Código: _____

Indicación: Sea breve en sus respuestas y justifiquelas concisamente. Excepto las preguntas de selección múltiple.

1. (20 points) **Calculadora**
   Dada la siguiente expresión:

   $$(R + 8)/3S + (R - 3S) + S$$

   Responda:

   a) ¿Cuál es el valor de la expresión anterior?

   b) ¿Cuál es el valor al final de la evaluación del registro de memoria?

   c) Muestre el árbol abstracto sintáctico de la anterior expresión.

2. (20 points) **BNF - Backus Naur Form**

   La siguiente es una versión modificada en BNF de la gramática de aritmética con memoria. Esta modificación permite más de una memoria.

   $$\begin{array}{rcl}
   \langle \text{Prog} \rangle & ::= & \langle \text{Expr} \rangle \text{ eof} \\
   \langle \text{Expr} \rangle & ::= & \langle \text{Expr} \rangle + \langle \text{Term} \rangle \\
   & | & \langle \text{Expr} \rangle - \langle \text{Term} \rangle \\
   & | & \langle \text{Term} \rangle \\
   \langle \text{Term} \rangle & ::= & \langle \text{Term} \rangle \times \langle \text{Storable} \rangle \\
   & | & \langle \text{Term} \rangle / \langle \text{Storable} \rangle \\
   & | & \langle \text{Storable} \rangle \\
   \langle \text{Storable} \rangle & ::= & \langle \text{Factor} \rangle S \text{ number} \mid \langle \text{Factor} \rangle \\
   \langle \text{Factor} \rangle & ::= & \text{number} \mid R \text{ number} \mid (\langle \rangle)
   \end{array}$$

   a) Muestre la derivación más a la izquierda que produzca la cadena similar al punto anterior y su correspondiente árbol de derivación.

   b) Muestre la derivación más a la derecha que produzca la cadena similar al punto anterior y su correspondiente árbol de derivación.

3. (20 points) **EWE**
   Escriba un programa en el lenguaje de programación EWE que lea tres números y muestre cual es el mayor de los tres.

4. (20 points) **Gramáticas**

   Considere la siguiente gramática:

$$S \rightarrow +SS \mid *SS \mid a$$

   a) Demuestre cómo se puede generar la cadena `+a**aaa` con esta gramática.

   b) ¿Qué lenguaje genera esta gramática? Explique la respuesta.

Solution:

1. La deriviación

$$\begin{aligned} S &\Rightarrow +SS \\ &\Rightarrow +aS \\ &\Rightarrow +a*SS \\ &\Rightarrow +a**SSS \\ &\Rightarrow +a**aSS \\ &\Rightarrow +a**aaS \\ &\Rightarrow +a**aaa \end{aligned}$$

   muesta que la cadena se puede generar.

2. El lenguaje es el de las expresiones aritméticas con prefijos.

5. (20 points) **Historia del lenguajes de programación**
   Responsa las siguientes preguntas de los siguientes lenguajes de programación basada en la lectura del capítulo del libro de M. Gabbrielli y S. Martini sobre la historia de los lenguajes de programación.

   a) ¿Clase de lenguajes que marcarón la influencia de los años 80?

   b) La sintaxis del lenguaje de programación ADA, ¿está basada en que lenguaje de programación?

   c) ¿Cuál es el significado del acrónimo de LISP?

   d) ¿Quién fue el creador del lenguaje de programación FORTRAN?

   e) ¿Qué significa el acrónimo de ML?

ST0244 Lenguajes de programación 031 **Universidad EAFIT**
Primer Parcial **DIS**
Juan Francisco Cardona McCormick **27 de febrero de 2014**

Nombre: _____
Código: _____

Indicación: Sea breve en sus respuestas y justifiquelas concisamente. Excepto las preguntas de selección múltiple.

1. (25 points) **Gramáticas independientes de contexto**
   La siguiente es la gramática de un lenguaje de bloques $G_{\text{Bloques}} = (V, \Sigma, P, S)$ donde:

$$
\begin{aligned}
V &= \{L, LE, E\} \\
\Sigma &= \{\ '[',\ ']',\ a,\ b,\ ',' \ \} \\
P &= \{\ L \rightarrow\ '[' \ LE \ ']' \ \mid\ '[' \ M \ ']', \\
&\quad\ M \rightarrow\ \epsilon \ \mid\ '[' \ M \ ']' \\
&\quad\ LE \rightarrow\ a \ \mid\ b \ \mid\ L \ \mid\ a \ ',' \ LE \ \mid\ b \ ',' \ LE \ \mid\ L \ ',' \ LE \ \} \\
S &= L
\end{aligned}
$$

   a) Muestre la derivación más a la izquierda que produzca la cadena siguiente cadena $[a, [[]], [b]]$.
   b) Muestre la derivación más a la derecha que produzca la cadena similar $[a, [[]], [b]]$.

Solution:

a) Derivación más a la izquierda:

$$
\begin{aligned}
L &\Rightarrow\ '[' \ LE ']' \\
&\Rightarrow\ '[' \ a \ ',' \ LE \ ']' \\
&\Rightarrow\ '[' \ a \ ',' \ \ L \ ',' \ LE \ ']' \\
&\Rightarrow\ '[' \ a \ ',' \ \ '[' \ M \ ']' \ \ ',' \ LE \ ']' \\
&\Rightarrow\ '[' \ a \ ',' \ \ '[' \ '[' \ M \ ']' \ ']' \ \ ',' \ LE \ ']' \\
&\Rightarrow\ '[' \ a \ ',' \ \ '[' \ '[' \ \epsilon \ ']' \ ']' \ \ ',' \ LE \ ']' \\
&\Rightarrow\ '[' \ a \ ',' \ \ '[' \ '[' \ ']' \ ']' \ \ ',' \ L \ ']' \\
&\Rightarrow\ '[' \ a \ ',' \ \ '[' \ '[' \ ']' \ ']' \ \ ',' \ '[' \ LE \ ']' \ ']' \\
&\Rightarrow\ '[' \ a \ ',' \ \ '[' \ '[' \ ']' \ ']' \ \ ',' \ '[' \ b \ ']' \ ']'
\end{aligned}
$$

b) Derivación más a la derecha:

$$
\begin{aligned}
L &\Rightarrow\ '[' \ LE ']' \\
&\Rightarrow\ '[' \ a \ ',' \ LE \ ']' \\
&\Rightarrow\ '[' \ a \ ',' \ \ L \ ',' \ LE \ ']' \\
&\Rightarrow\ '[' \ a \ ',' \ \ L \ ',' \ L \ ']' \\
&\Rightarrow\ '[' \ a \ ',' \ \ L \ ',' \ '[' \ LE \ ']' \ ']' \\
&\Rightarrow\ '[' \ a \ ',' \ \ L \ ',' \ '[' \ b \ ']' \ ']' \\
&\Rightarrow\ '[' \ a \ ',' \ \ '[' \ M \ ']' \ \ ',' \ '[' \ b \ ']' \ ']' \\
&\Rightarrow\ '[' \ a \ ',' \ \ '[' \ '[' \ M \ ']' \ ']' \ ]',' \ '[' \ b \ ']' \ ']' \\
&\Rightarrow\ '[' \ a \ ',' \ \ '[' \ '[' \ \epsilon \ ']' \ ']' \ ]',' \ '[' \ b \ ']' \ ']'
\end{aligned}
$$

2. (25 points) **EWE**
Escriba un programa en EWE que lea uno número entero a e indique cuantos digitos decimales está compuesto.

Ejemplos:

$$\begin{array}{lcll} a & = & 12343 & \rightarrow 5 \\ a & = & 232 & \rightarrow 3 \\ a & = & 1344 & \rightarrow 4 \\ a & = & 1 & \rightarrow 1 \\ a & = & 0 & \rightarrow 1 \end{array}$$

Nota: Recuerde que $a \,\%b$ retorna un valor $0 \ldots (b-1)$ y que la división es una división entera, es decir que si $1234/10$ es igual a 123.

Solution:

```
# Cuenta el número de digitos que tiene un número entero
cero := 0
uno  := 1
diez := 10
ndig := 0
readInt(a)
loop: a := a / diez
ndig := ndig + uno
if a <> cero then goto loop
writeInt(ndig)
halt
equ cero M[0]
equ uno M[1]
equ diez M[2]
equ ndig M[3]
equ a M[4]
```

3. (25 points) **Ambigüedad**
Mostrar la ambigüedad en la siguiente gramática independiente de contexto G_3:

$$G_3 = (V = \{Z, Y\}, \Sigma = \{z, y, x, w\}, P = \{Z \to ZYZ \mid z, Y \to y \mid x \mid w, Z)$$

Solution: Observamos que la gramática tiene un problema de ambigüedad en la primera producción, dado que esta utilizando el no-terminal como Y operador y Z no tiene un orden de derivación, se sabe que tiene problemas de ambigüedad. Observemos la cadena *abaca*, genera a través de dos derivaciones distintas dos árboles de derivación diferentes.

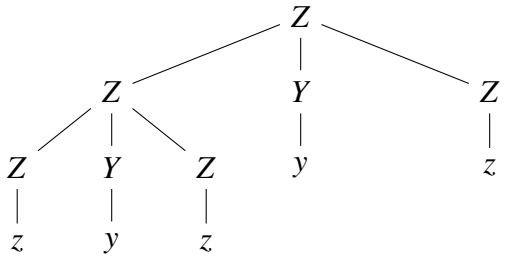
Primera derivación:

$$\begin{aligned} Z &\Rightarrow ZYZ \\ &\Rightarrow ZYZYZ \\ &\Rightarrow zYZYZ \\ &\Rightarrow zyZYZ \\ &\Rightarrow zyzYZ \\ &\Rightarrow zyzyZ \\ &\Rightarrow zyzyz \end{aligned}$$

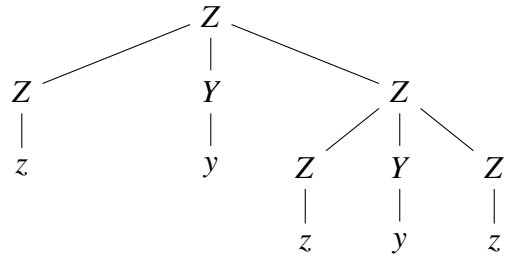
Segunda derivación:

$$\begin{aligned} Z &\Rightarrow ZYZ \\ &\Rightarrow ZYZYZ \\ &\Rightarrow ZYZYz \\ &\Rightarrow ZYZyz \\ &\Rightarrow ZYzyz \\ &\Rightarrow Zyzyz \\ &\Rightarrow zyzyz \end{aligned}$$

Se obtiene el siguiente árbol:



Se obtiene el siguiente árbol:



Esto genera dos árboles de derivación distintos para la misma cadena por lo tanto la gramática es ambigua.

4. (25 points) **Implementación de los lenguajes de programación**

   ¿Por qué interpretar un programa es preferido sobre la compilación de un programa?

Solution: Los programas interpretados tiene varias ventas sobre los programas compilados:

a) El ciclo de escribir y poner a funcionar un programa es un ciclo muy corto que los lenguajes compilados.

b) La sintaxis es más flexible.

c) Permite corregir más fácilmente los errores de los programas.

d) En general, son independientes de la plataforma.

ST0244 Lenguajes de programación 032 **Universidad EAFIT**
Primer Parcial **DIS**
Juan Francisco Cardona McCormick **27 de febrero de 2014**

Nombre: _____

Código: _____

Indicación: Sea breve en sus respuestas y justifiquelas concisamente. Excepto las preguntas de selección múltiple.

1. (25 points) **Gramáticas independientes de contexto**

   La siguiente es la gramática de un lenguaje de bloques $G_{\text{bloques}} = (V, \Sigma, P, S)$ donde:

$$
\begin{aligned}
V &= \{L, LE, E\} \\
\Sigma &= \{\ '[',\ ']',\ a,\ b,\ ',' \ \} \\
P &= \{L \rightarrow\ '[' \ LE \ ']' \ \mid\ '[' \ M \ ']', \\
&\quad\ M \rightarrow\ \epsilon \ \mid\ '[' \ M \ ']' \\
&\quad\ LE \rightarrow\ E \ \mid\ E \ ',' \ LE, \\
&\quad\ E \rightarrow\ a \ \mid\ b \ \mid\ L\ \} \\
S &= L
\end{aligned}
$$

   a) Muestre la derivación más a la izquierda que produzca la cadena siguiente cadena $[a, [[]], []]$.

   b) Muestre la derivación más a la derecha que produzca la cadena similar $[a, [[]], []]$.

Solution:

a) Derivación más a la izquierda:

$$
\begin{aligned}
L &\Rightarrow\ '[' \ LE \ ']' \\
&\Rightarrow\ '[' \ E \ ',' \ LE \ ']' \\
&\Rightarrow\ '[' \ a \ ',' \ LE \ ']' \\
&\Rightarrow\ '[' \ a \ ',' \ E \ ',' \ LE \ ']' \\
&\Rightarrow\ '[' \ a \ ',' \ L \ ',' \ LE \ ']' \\
&\Rightarrow\ '[' \ a \ ',' \ '[' \ '[' \ M \ ']' \ ']' \ ',' \ LE \ ']' \\
&\Rightarrow\ '[' \ a \ ',' \ '[' \ '[' \ \epsilon \ ']' \ ']' \ ',' \ LE \ ']' \\
&\Rightarrow\ '[' \ a \ ',' \ '[' \ '[' \ ']' \ ']' \ ',' \ E \ ']' \\
&\Rightarrow\ '[' \ a \ ',' \ '[' \ '[' \ ']' \ ']' \ ',' \ L \ ']' \\
&\Rightarrow\ '[' \ a \ ',' \ '[' \ '[' \ ']' \ ']' \ ',' \ '[' \ M \ ']' \ ']' \\
&\Rightarrow\ '[' \ a \ ',' \ '[' \ '[' \ ']' \ ']' \ ',' \ '[' \ \epsilon \ ']' \ ']'
\end{aligned}
$$

b) Derivación más a la derecha:

$$
\begin{aligned}
L &\Rightarrow \text{'[' } LE \text{ ']'} \\
&\Rightarrow \text{'[' } E \text{ ',' } LE \text{ ']'} \\
&\Rightarrow \text{'[' } a \text{ ',' } LE \text{ ']'} \\
&\Rightarrow \text{'[' } a \text{ ',' } E \text{ ',' } LE \text{ ']'} \\
&\Rightarrow \text{'[' } a \text{ ',' } E \text{ ',' } E \text{ ']'} \\
&\Rightarrow \text{'[' } a \text{ ',' } E \text{ ',' } L \text{ ']'} \\
&\Rightarrow \text{'[' } a \text{ ',' } E \text{ ',' '[' } M \text{ ']' ']'} \\
&\Rightarrow \text{'[' } a \text{ ',' } E \text{ ',' '[' } \epsilon \text{ ']' ']'} \\
&\Rightarrow \text{'[' } a \text{ ',' } L \text{ ',' '[' ']' ']'} \\
&\Rightarrow \text{'[' } a \text{ ',' '[' } M \text{ ']' ',' '[' ']' ']'} \\
&\Rightarrow \text{'[' } a \text{ ',' '[' '[' } M \text{ ']' ']' ',' '[' ']' ']'} \\
&\Rightarrow \text{'[' } a \text{ ',' '[' '[' } \epsilon \text{ ']' ']' ',' '[' ']' ']'}
\end{aligned}
$$

2. (25 points) **EWE**
Escriba un programa en EWE que lea dos números enteros positivos: a y b donde b es un número de un sólo dígito. El programa debe mirar en a cuanto dígitos b contiene:

Ejemplo 1:

$$\begin{aligned} a &= 12343 \\ b &= 3 \\ |\, a \,|_b &= 2 \end{aligned}$$

Ejemplo 2:

$$\begin{aligned} a &= 12343 \\ b &= 1 \\ |\, a \,|_b &= 1 \end{aligned}$$

Ejemplo 3:

$$\begin{aligned} a &= 12343 \\ b &= 6 \\ |\, a \,|_b &= 0 \end{aligned}$$

Nota: Recuerde que $a\,\%b$ retorna un valor $0 \ldots (b-1)$ y que la división es una división entera, es decir que si $1234/10$ es igual a 123.

Solution:

```
# Cuenta el número de digitos b que tiene un número entero a
cero := 0
uno  := 1
diez := 10
ndigbea := 0
readInt(a)
readInt(b)
loop: dig := a % diez
a := a / diez
if b <> dig then goto end
ndigbea := ndigbea + uno
end: if a <> cero then goto loop
writeInt(ndigbea)
halt
equ cero M[0]
equ uno M[1]
equ diez M[2]
equ ndigbea M[3]
equ a M[4]
equ b M[5]
equ dig M[6]
```

3. (25 points) **Gramáticas**
   Considere la siguiente gramática:

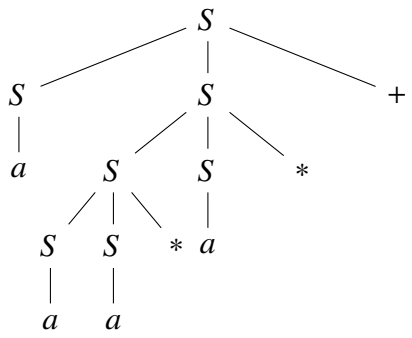
$$S \rightarrow SS+ \mid SS* \mid a$$

Considere la siguiente derivaación de la gramática:

$$\begin{aligned} S &\Rightarrow SS+ \\ &\Rightarrow aS+ \\ &\Rightarrow aSS*+ \\ &\Rightarrow aSS*S*+ \\ &\Rightarrow aaS*S*+ \\ &\Rightarrow aaa*S*+ \\ &\Rightarrow aaa*a*+ \end{aligned}$$

Construya su correspondiente árbol de análisis (derivación).

Solution:



4. (25 points) **Implementación de los lenguajes de programación**

   ¿Por que compilar un programa es preferido sobre la interpretación de un programa?

Solution:

a) La sintáxis es más estricta, no permite errores extraños en el tiempo de compilación.

b) Una gran parte de los errores son detectados en tiempo de compilación.

c) Su código corre más rápido que los interpretados.

Nombre: _____

Código: _____

Indicación: Sea breve en sus respuestas y justifiquelas concisamente. Excepto las preguntas de selección múltiple.

1. (20 points) **De UML a C++**

   En la siguiente figura 1 hay un diagrama de clases. Defina la interfaz de las todas clases en C++, y muestre la implementación de las clases `CuentaCorriente` y `Cuenta`. Recuerde que las funciones `obtenerSaldo`, `consignación` y `retiro` son funciones polimórficas. La diferencia entre una cuenta de ahorros y la cuenta corriente es que la segunda permite hacer retiros mayores a los que se tiene consignados, pero tiene un tope máximo en hacer este retiro.

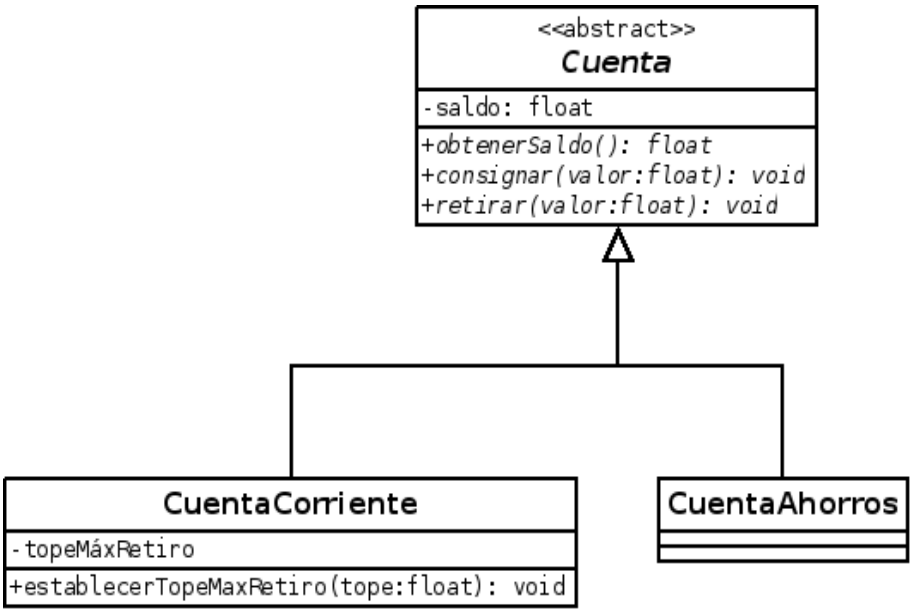


Figura 1: Diagrama de una jerarquía de clases

Solution: A continuación se muestra la interfaz de todas las clases involucradas:

```
1 class Cuenta {
2 public:
3   Cuenta();
4   Cuenta(float saldo);
5   virtual ~Cuenta();
6   virtual float obtenerSaldo() const;
7   virtual void consignar(float valor);
```

```
  virtual void retirar(float valor);
private:
  float saldo;
};

class CuentaAhorros : public Cuenta {
public:
  CuentaAhorros();
  CuentaAhorros(float saldo);
  ~CuentaAhorros();
};

class CuentaCorriente : public Cuenta {
public:
  CuentaCorriente();
  CuentaCorriente(float saldo, float topeMaxRetiro);
  virtual ~CuentaCorriente();
  void establecerTopeMaxRetiro(float topeMaxRetiro);
  virtual void retirar(float valor);
private:
  float topeMaxRetiro;
};
```

La implementación de las clases `CuentaCorriente` y `Cuenta`:

```
Cuenta::Cuenta() : saldo(0.0f) { }

Cuenta::Cuenta(float saldo) : saldo(saldo) { }

Cuenta::~Cuenta() { }

float Cuenta::obtenerSaldo() const {
  return saldo;
}

void Cuenta::consignar(float valor) {
  saldo += valor;
}

void Cuenta::retirar(float valor) {
  saldo -= valor;
}

CuentaCorriente::CuentaCorriente()
  : Cuenta() , topeMaxRetiro(0.0f) { }

CuentaCorriente::CuentaCorriente(float saldo, float topeMaxRetiro)
  : Cuenta(saldo), topeMaxRetiro(topeMaxRetiro) { }

```

```
25 CuentaCorriente::~CuentaCorriente() { }
26
27 void
28 CuentaCorriente::retirar(float valor) {
29
30   if (valor >= obtenerSaldo()) {
31     if (valor <= obtenerSaldo() + topeMaxRetiro) {
32       Cuenta::retirar(valor);
33     }
34   }
35 }
36
37 void
38 CuentaCorriente::establecerTopeMaxRetiro(float topeMaxRetiro) {
39
40   this->topeMaxRetiro = topeMaxRetiro;
41 }
```

2. (20 points) **Memoria**

El siguiente programa en C++:

```
1  int* x;
2
3  int f(int &a) {
4      x = &a;
5
6      return *x * a;
7  }
8
9  int g(int b) {
10     int c;
11     return f(c) + b;
12 }
13
14 int
15 main() {
16     int w;
17     int *p = &w;
18
19     cout << g(*p) << endl;
20     return 0;
21 }
22
```

Define variables en diferentes partes de la memoria. Según el modelo de memoria presentado en la figura 2, muestre en qué parte están ubicadas las variables e *explique* por que están allí.

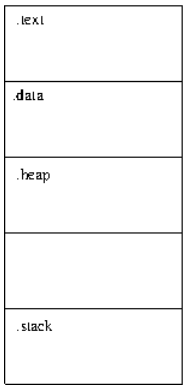


Figura 2: Modelo de memoria simplificad de los programas en C++

Solution: La variable definida en la línea 1 es un puntero a entero que es global por lo tanto esta definido en la región de `.data`.

La variable `a` definida en la línea 3 es referencia, la referencia es local, pero cuando es invocada depende del valor invocado. Toda variable local está definda en la región de `.stack`.
En la líneas 9 y 10 están definida dos variables locales `b` y `c`, al ser locales están definidas en la región `.stack`.
En las línas 16 y 17 están definidas dos variables que son locales a la función main, la primera es un entero `w`, se encuenta definida en la región del `.stack`. La segunda, también se encuentra definida en el `.stack`.

3. (20 points) **Punteros, referencias y paso de parámetros**

El siguiente programa en C++:

```
1  void f(int* a, int * b) {
2     int c = *a, d = *b;
3     a = new int;
4     b = new int;
5     *a = d;
6     *b = c;
7  }
8
9  int
10 main() {
11   int x = 10, y = 20;
12
13   f(&x,&y);
14   cout << "x: " << x << " y: " << y << endl;
15   f(&y, &x);
16   cout << "x: " << x << " y: " << y << endl;
17 }
18
```

¿Qué imprime en la salida el anterior programa y por qué?

Solution: La salida del anterior programa es:

```
10 20
10 20
```

La función `f` luce como una función de intercambio, donde el valor del puntero *a* se guarda en *c* y el valor del puntero *b* se guarda en *d*, pero dentro de la función se cambia los punteros *a* y *b* por nuevas posiciones de memoria, la anteriores no se cambian, y en estas nuevas posiciones se guardan los valores anteriores al nuevo *a* se guarda *d* y al nuevo *b* se guarda *c*, pero los valores de los punteros anteriores no se tocan para nada, por lo tanto los valores originales no cambian.

4. (20 points) **Punteros, referencias y paso de parámetros**

El siguiente programa es un ejemplo del uso de punteros y referencias:

```
1  void f(int a, int& b) {
2    a = a + b;
3    b = a - b;
4    a = a - b;
5  }
6
7  void g(int* a, int b) {
8    *a = *a + b;
9    b  = *a - b;
10   *a = *a - b;
11 }
12
13 int main() {
14   int x = 1, y = 2;
15   f(x,y);
16   cout << "x: " << x << " y: " << y << endl;
17   g(&x,y);
18   cout << "x: " << x << " y: " << y << endl;
19   return 0;
20 }
```

¿Qué imprime en la salida el anterior programa y por qué?

Solution: La salida del anterior programa es la siguiente:

```
1 1
1 1
```

En primer lugar, observemos la primera función, tiene dos parámetros `a` y `b`. El primer parámetro es un paso por valor y el segundo es un parámetro por referencia. La función f hace un intercambio entre los valores de `a` y `b`, pero solo el segundo parametro se intercambia el primero se va mantener. Entonces, cuando en la línea 15 de invoca a la función `f` solamente el valor de `y` será intercambiado con el valor `x`, pero `x` no cambia por lo tanto al final de la invocación: `x = 1` y `y = 1`.
La función `g` es también un función de intercambio, pero solamente cambio el valor del primer parametro `a` y no del segundo, por que el primero es un puntero a una posición de memoria y el segundo es un parámetro por valor. Al ser invocada con los valores actuales de `x` y `y`, `1` y `1` respectivamente, solamente "cambia el valor del primer con el segundo. por lo tanto el resultado es: `x = 1` y `y = 1`.

5. (20 points) **Punteros, referencia e invocación de parámetros**
   El siguiente programa es un ejemplo del uso de punteros y referencias:

```
1  int incr(int a, int& b) { b += a;  return b;}
2
3  int decr(int& a, int b) { b -= a;   return b;}
4
5  void eval(int* a, int& b) {
6    *a = incr(*a, b);
7    b = decr(b, *a);
8  }
9
10 int main() {
11    int x = 10, y = 20;
12
13    eval(&y, x);
14    cout << "x: " << x << " y: " << y << endl;
15 }
```

¿Qué imprime en la salida el anterior programa y por qué?

Solution: En `main` se declaran dos variables $\texttt{x}_{main}$ y $\texttt{y}_{main}$, con los valores de `10` y `20` respectivamente. La invocación a eval enlaza las variables locales $\texttt{a}_{eval}$ con la dirección de $\texttt{y}_{main}$ y a la variable $\texttt{b}_{eval}$ con la variable $\texttt{x}_{main}$ (por referencia).
La primera instrucción de la función `eval` (línea 6) deja el resultado de la invocación de la función `incr` en la variable $\texttt{a}_{eval}$. Esta función tiene dos parámetros el primero $\texttt{a}_{incr}$ es paso por valor es decir que tiene una copia del valor de $\texttt{a}_{eval}$; y el segundo parámetro $\texttt{b}_{incr}$ es una referencia al valor de $\texttt{b}_{eval}$, que es es la referencia a $\texttt{x}_{main}$. Cuando se ejecuta la única instrucción de `incr`, los valores de $\texttt{b}_{incr}$ queda en `30` por lo tanto el valor de $\texttt{b}_{eval}$ y de $\texttt{x}_{main}$ quedan en `30`. Al retornar de `incr` contenido del apuntador de $\texttt{a}_{eval}$ queda en `30` lo que implica que el valor de $\texttt{y}_{main}$ también queda en `30`.
Luego se ejecuta la segunda instrucción de la función `eval` (línea 7) que deja el resultado de la función `decr` en la variable $\texttt{b}_{eval}$.Esta función tiene dos parámetros: el primero $\texttt{a}_{decr}$ es una referencia es decir está ligado a $\texttt{b}_{eval}$ y también a $\texttt{s}_{main}$; el segundo parámetro $\texttt{b}_{decr}$es un copia del valor de $\texttt{a}_{eval}$. El resultado de ejecutar la única instrucción de `decr` solo cambia el contenido de $\texttt{b}_{decr}$ que tiene como valor a `0`. Este resultado es retornado a la variable $\texttt{b}_{eval}$ que hace que tome el valor de 30, por ende el valor $\texttt{x}_{main}$ también queda en `0`.
De lo anterior decimos que se imprimime:

```
x: 0 y: 30
```

Nombre: _____

Código: _____

Indicación: Sea breve en sus respuestas y justifiquelas concisamente. Excepto las preguntas de selección múltiple.

1. (20 points) **De UML a C++**

   En la siguiente figura 1 hay un diagrama de clases. Defina la interfaz de las todas clases en C++, y muestre la implementación de las clases `Figure` y `Rectangle`. Recuerde que las funciones `compArea` y `graphic` son funciones polimórficas.

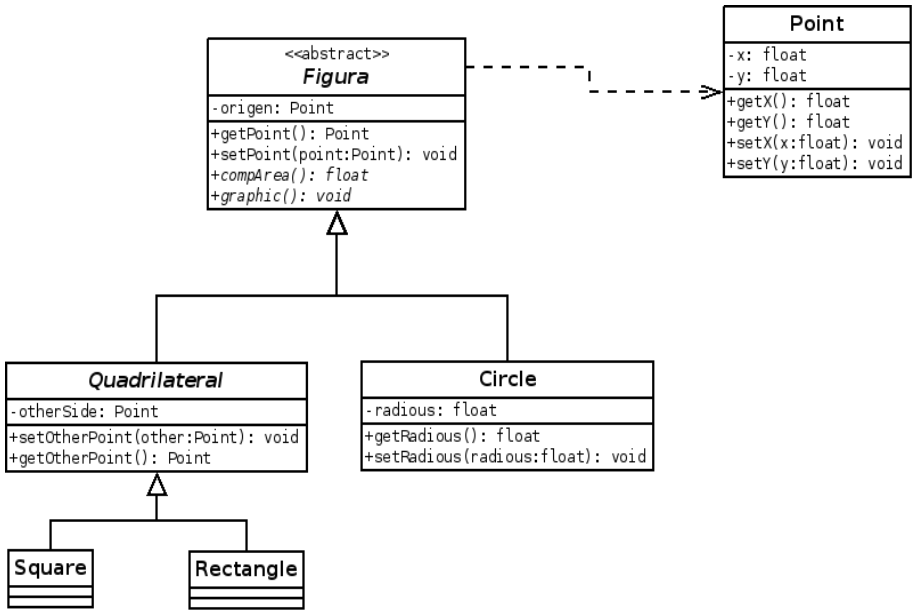


Figura 1: Diagrama de una jerarquía de clases

Solution: La interfaz de todas las clases:

```
1 class Point {
2 public:
3   Point();
4   Point(float x, float y);
5   ~Point();
6   float getX() const;
7   float getY() const;
8   void setX(float x);
9   void setY(float y);
```

```
private:
  float x;
  float y;
};

class Figure {
public:
  Figure(Point origen);
  virtual ~Figure();
  Point getPoint() const;
  void setPoint(Point point);
  virtual float compArea() = 0;
  virtual void graphic() = 0;
private:
  Point origen;
};

class Quadrilateral : public Figure {
public:
  Quadrilateral(Point origen, Point otherSide);
  ~Quadrilateral();
  Point getOtherPoint() const;
  void setOtherPoint(Point other);
private:
  Point otherSide;
};

class Circle : public Figure {
public:
  Circle(Point origen, float radious);
  ~Circle();
  float getRadious() const;
  void setRadious(float radious);
private:
  float radious;
};

class Rectangle : public Quadrilateral {
public:
  Rectangle(Point origen, Point otherSide);
  ~Rectangle();
  virtual float compArea();
  virtual void graphic();
};

class Square : public Quadrilateral {
public:
  Square(Point origen, Point otherSide);
  ~Square();
  virtual float compArea();
  virtual void graphic();
```

```
};
```

La implementación de las clases `Figure` y `Rectangle`

```
Figure::Figure(Point origen) : origen(origen) { }

Figure::~Figure() { }

Point Figure::getPoint() const {
  return origen;
}

void Figure::setPoint(Point origen) {
  this->origen = origen;
}

Rectangle::Rectangle(Point origen, Point otherSide) :
  Quadrilateral(origen, otherSide) { }

Rectangle::~Rectangle() { }

float Rectangle::compArea() {
  Point origen = this->getPoint();
  Point otherSide = this->getOtherPoint();
  return (fabs(origen.getX() - otherSide.getX())) * (fabs(origen.getY()
    - otherSide.getY()));
}

void Rectangle::graphic() {
}
```

2. (20 points) **Memoria**

El siguiente programa en C++:

```
1  int x;
2
3  int f(int x) {
4      ...
5  }
6
7  int main() {
8      int x;
9      f(x);
10     int *y = new int;
11     return 0;
12 }
13
```

Define variables en diferentes partes de la memoria. Según el modelo de memoria presentado en la figura 2, muestre en que parte están ubicadas las variables e señale por que están ahí.

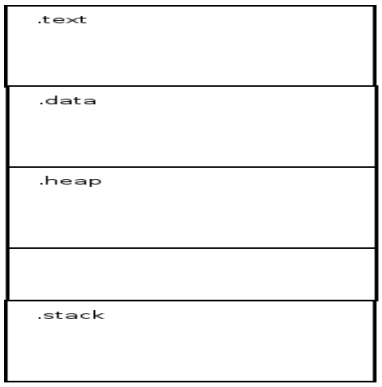


Figura 2: Modelo de memoria simplificad de los programas en C++

Solution: En la línea 1 se observa la declaración de la variable `x` esta variable es global, por lo tanto debe estar ubicada en la región `.data`, donde se declarán las variables globales.
En las líneas 3 a 5 se declara una función `f` que tiene como argumento a la variable `x`. Esta es una variable local declarada dentro de una función por lo tanto al ser una variable local, es declara en la region `.stack`.
En la líneas 7 a 12 se declara un función `main`, en ella se declaran varias variables: `x` y `y`, ambas son variables locales por lo tanto ambas están definidas en la región de `.stack`.

3. (20 points) **Punteros, referencias y paso de parámetros**

El siguiente programa en C++:

```
1  void f(int a, int *b) {
2     int t;
3     t = a;
4     a = *b;
5     *b = t;
6  }
7
8  int main() {
9    int a = 10;
10   int b = 20;
11
12   f(y, &x);
13   cout << a << " " << b << endl;
14   return 0;
15 }
16
```

¿Qué imprime en la salida el anterior programa y por qué?

Solution: El programa original tenía dos erratas que ya se encuentra corregidas. La salida del programa es:

```
10 10
```

Por que el programa de intercambio aunque hace el intercambio internamente dentro de la función `f`, el primer parámetro es por valor y no por referencia, mientras que el segundo si es por referencia. Solamente cambia por fuera al segundo valor.

4. (20 points) **Punteros, referencias y paso de parámetros**

El siguiente programa es un ejemplo del uso de punteros y referencias:

```
void swap(int& a, int* b) {
  a = a + *b;
  b = a - *b;
  a = a - *b;
}
```

La idea es realizar el intercambio entre dos tipos de datos sin utilizar una variable temporal. Pero el anterior programa tiene un error, podría corregirlo sin cambiar la interfaz de la función.

Solution: Se observa que la idea es utilizar la variable `a` como un acumulador para guardar los valores de `a` y `*b`, pero el problema se presenta en la segunda asignación, falta guardar el contenido en la posición apuntada por `*b`, por lo tanto la solución es:

```
void swap(int& a, int *b) {
  a  = a + *b;
  *b = a - *b;
  a  = a - *b;
}
```

5. (20 points) **Calculadora**
Suponga que la operación de residuo se ha añadido a la definición de la calculadora, añadiendo el opererador `%` como la operación de residuo de la calculadora. La siguiente es la gramática modificada $LL(1)$:

$$
\begin{array}{rcl}
Prog & \rightarrow & Expr \text{ EOF} \\
Expr & \rightarrow & Term\ RestExpr \\
RestExpr & to & +\ Term\ RestExpr \mid -\ Term\ RestExpr \mid \epsilon \\
Term & \rightarrow & Storable\ RestTerm \\
RestTerm & \rightarrow & *\ Storable\ RestTerm \mid /\ Storable\ RestTerm \mid \%\ Storable\ RestTerm \mid \epsilon \\
Storable & \rightarrow & Factor\ \text{S} \mid Factor \\
Factor & \rightarrow & \text{number} \mid textR \mid '('\ Expr\ ')'
\end{array}
$$

Indique con precisión: ¿Qué partes de la calculadora deben ser modificadas para aceptar este nuevo cambio?

Solution: En primer lugar al añadir un nuevo lexema, se necesario modificar el fichero que contiene la información de la clase `Token` añandiendo un nuevo tipo de token: `mod`[1].
Luego se debe modificar el analizador léxico (*scanner*) para que reconozca el nuevo lexema y cree una instancia de la clase `Token` con este nuevo tipo.
El siguiente punto donde se debe modificar es la clase `AST`, se debe añadir una nueva subclase a la clase `BinaryTree`, llamada `ModNode`[2]. A estea clase se le altera el método `evaluate` para que realize la operación módulo.

ST0244 Lenguajes de programación 032 **Universidad EAFIT**
Segundo parcial **DIS**
Juan Francisco Cardona McCormick **3 de abril de 2014**

Nombre: _____
Código: _____

1. (25 points) **Cadenas de carácteres**

   Escriba una función `concatenar`, esta función recibe dos cadenas de caracteres: `s` y `t`. La cadena `s` tiene espacio suficiente para recibir la cadena `t`. La siguiente es la signatura (*firma*) de la función:

```c
char* concantenar(char *s, const char *t);
```

   El valor retornado es la dirección de inicio de la cadena resultante.

Solution:

```c
char* concatenar(char *s, const char *t) {

    int i;
    for (i = 0; s[i]; i++);
    int j;
    for (j = 0; t[j]; j++,i++) {
      s[i] = t[j];
    }
    s[i] = '\0';
    return s;
}
```

2. (25 points) **Tipos de datos en C++ y argumentos de funciones**

   La función `promedio`, toma como único argumento un apuntador a entero, que representa un arreglo de enteros y retorna el valor promedio[1] de todos los elementos del arreglo. El valor esperado es un entero. El siguiente código es la signatura (*prototipo* o *firma*) de la función.

```
1 int promedio(const int* arreglo);
```

   ¿Son suficientes los parámetros para que la función pueda ser llevada a cabo? Si su respuesta es negativa: reescriba la definición para que ella logre su cometido.

   Solution: Los parámetros no son suficientes debido a que no se sabe el número total de valores de tiene el arreglo.

```
1 int promedio(const int* arreglo, const int longitud) {
2   int suma = 0;
3
4   for (int i = 0; i < longitud; i++) {
5     suma = suma + arreglo[i];
6   }
7   return (suma / longitud);
8 }
```

[1] Un promedio es igual a la suma de todos los elementos divido el número total de elementos

3. (25 points) **Punteros**

¿Qué puede pasar en el siguiente programa? Realice un análisis exhaustivo.

```
1  #include <iostream>
2
3  using namespace std;
4
5  int main() {
6      int *j;
7      int &r = *j;
8
9      for (; r < 10; r++)
10         cout << "Hola Mundo" << endl;
11
12     return 0;
13 }
```

Solution: En la línea 6 se observa la declaración de la variable `j` que es un tipo de dato puntero a entero, este puntero a entero no ha sido inicializado, por lo tanto puede estar apuntando a cualquier parte de memoria, válida o invalida. Si es inválida, el programa terminará inmediatamente debido a un acceso incorrecto. Si es válida, el valor puede ser menor o igual a diez o diferente. En el primer caso: el programa terminará, en el segundo imprimirá el mensaje un ciclo casi infinito.

4. (25 points) **Ámbito**

   Cual es la salida del siguiente programa en C++?

```
1  #include <iostream>
2
3  using namespace std;
4
5  int w;
6
7  int
8  f(int w) {
9    w = 5;
10   {
11     int w = ::w;
12     cout << "w= " << w << endl;
13   }
14 }
15
16 int
17 main() {
18   w = 3;
19
20   f(w);
21 }
```

Solution:

```
$ ./ST0244-2014-1-031-Parcial-02-Punto-04
w= 3
```

En la línea 18 el programa inicializa el valor de la variable `w` que es un valor global con 3. Luego llama a la función `f` que tiene una copia del valor global en su variable local `w`. Dentro de la función el valor local de `w` es puesto a 5 pero esto no afecta el valor global, luego el programa entra al bloque interno de la función `f` y se define una nueva variable `w` que es inicializada con el valor global `::w`, por lo tanto el valor de salida es `3`.

Nombre: _____
Código: _____

1. (25 points) **Referencias**

   En C++, Considere la siguiente función `aumentar`:

```
void aumentar(int &i, int &j, int k) {
    i = 10;
    j = 20;
    k = 30;
}

```

   ¿Cuál es la salida del siguiente fragmento de programa?

```
#include <iostream>

using namespace std;

int  main() {
  int i = 1;
  int j = 2;
  int k = 3;
  aumentar(k,j,k);
  cout << "El valor de i:" << i << " j: "  << j
       << " k: " << k << endl;
  return 0;
}

```

Solution: Al ejecutar el anterior programa se obtiene:

```
El valor de i: 1 j: 20 k: 10
```

La invocación original valor real `k` con el parametro formal `i` de la función `aumentar` y como es una referencia son el mismo objeto `k`. El segundo parámetro `j` con el parámetro formal `j` de la función `aumentar` y como este último parámetro es una referencia son el mismo objeto. El tercer parámetro es copiado con el valor del parámetro formal `k` al parámetro formal `k` de la función `aumentar`, no son el mismo objeto.

En la función la variable `i` modifica al parámetro real `k` en `main`, entonces `k` tiene el valor de 10. Igualmente, en la función la variable `j` modifica al parámetro real `j` en main, entonces `j` en main tiene el valor de 20. El último parámetro `k` de la función `aumentar` es un solo valor de copia.

2. (25 points) **Asignación**
   ¿Qué imprime el siguiente programa?

```
#include <iostream>

using namespace std;

int main() {
   bool bCentinela = false;
   int i = 0;

   while (bCentinela = true) {
     if (i < 10) i++;
     else bCentinela = false;
   }
   cout << "i:" << i << endl;
   return 0;
}
```

Solution:

El programa nunca va a terminar por que el ciclo en el `while` es una asignación que siempre va a ser verdadera por lo tanto se queda en un ciclo infinito y nunca llega a imprimir nada.

3. (25 points) **Punteros**

¿Qué puede pasar en el siguiente programa? Realice un análisis exhaustivo.

```
#include <iostream>

using namespace std;

int main() {
    int *i;

    for (*i = 0; *i < 10; (*i)++)
        cout << "Hola Mundo" << endl;

    return 0;
}
```

Solution: La variable `i` está definida como un puntero a entero, pero esta variable no ha sido asignada, tiene un valor desconocido allí generalmente esto causaría que el programa termine abrutamente por que no se sabe a que esta apuntando `i`.

En algunas circunstacias podría funcionar pero esto se debe al valor que apunta `i` puede ser un valor de memoria válido, pero podría causar otros efectos colaterales.

4. (25 points) **Ámbito**

¿Cuál es la salida del siguiente programa?

```
1  #include <iostream>
2
3  using namespace std;
4
5  int x;
6
7  void f(int &y) {
8    int x = ::x;
9    y += x;
10 }
11
12 int g(int& y) {
13   int x = ::x + 10;
14   y -= 10;
15   return y;
16 }
17
18 int main() {
19   int x = ::x;
20   f(x);
21   x = g(::x);
22   cout << "x: " << x
23        << "::x: " << ::x << endl;
24   return 0;
25 }
```

Solution: La salida del anterior programa es:

```
$ ./ST0244-2014-1-032-Parcial-02-Punto-04
x: -10::x: -10
```

El programa comienza en la instrucción `19`, allí se declara la variable `x` pero esta se inicializa con el valor de la variable global `::x`, esta última no se inicializa y es inicializada con el valor cero `0`. En la línea `20` se invoca a la función `f(x)` con el argumento de la variable `x` local. Esta función tiene el argumento una referencia en este caso a la variable local `x` en la función `main`. La variable interna `x` de la función main es inicializada con la variable global `::x`. Por lo tanto ambos tiene cero, cuando se incrementa la referencia `y` esta continua en cero. Por lo tanto esta invocación deja las variables tal como están.

En la línea 21 se llama a la función `g` con un argumento que es la variable global `::x`. En la función `g` tiene una referencia por argumento, por lo tanto el argumento `y` se refiere a la variable global `::x`. La variable local `x` de la función `g`. Su valor es iniciado con el valor

de la variable global `::x` más 10. El valor de `y` es decrementado en `10`, por lo tanto este valor de `::x`; y se retorna este valor que se asigna a la variable local `x` de main.

Nombre: _____
Código: _____

1. (20 puntos) **Cálculo lambda**

Evaluar las dos siguiente expresiones λ a su forma normal utilizando los dos mecanismos de evaluación: reducción de orden normal y reducción de orden aplicativo en cada una de las expresiones lambda.

a) $(\lambda s.(s\ s))((\lambda s.(s\ s))(\lambda s.(s\ s)))$
b) $(\lambda s.(s\ s))(\lambda x.x)(\lambda s.(s\ s))$

Solution:

a)
- Reducción de orden normal:

$$\begin{aligned}(\lambda s.(s\ s))((\lambda s.(s\ s))(\lambda s.(s\ s))) &\Rightarrow ((\lambda s.(s\ s))(\lambda s.(s\ s)))((\lambda s.(s\ s))(\lambda s.(s\ s)))\\ &\Rightarrow ((\lambda s.(s\ s))(\lambda s.(s\ s)))((\lambda s.(s\ s))(\lambda s.(s\ s)))\\ &\Rightarrow ((\lambda s.(s\ s))(\lambda s.(s\ s)))((\lambda s.(s\ s))(\lambda s.(s\ s)))\end{aligned}$$

La reducción normal no termina y sigue con las mismas expresiones hasta el infinito.

- Reducción de orden aplicativo:

$$\begin{aligned}(\lambda s.(s\ s))((\lambda s.(s\ s))(\lambda s.(s\ s))) &\Rightarrow (\lambda s.(s\ s))(((\lambda s.(s\ s))(\lambda s.(s\ s)))((\lambda s.(s\ s))(\lambda s.(s\ s))))\\ &\Rightarrow (\lambda s.(s\ s))(((\lambda s.(s\ s))(\lambda s.(s\ s)))((\lambda s.(s\ s))(\lambda s.(s\ s))))\\ &\Rightarrow (\lambda s.(s\ s))(((\lambda s.(s\ s))(\lambda s.(s\ s)))((\lambda s.(s\ s))(\lambda s.(s\ s))))\end{aligned}$$

La redución en orden aplicativo no termina y sigue con las mismas expresiones hasta el infinito.

b)
- Reducción de orden normal:

$$\begin{aligned}(\lambda s.(s\ s))(\lambda x.x)(\lambda s.(s\ s)) &\Rightarrow ((\lambda x.x)(\lambda x.x))(\lambda s.(s\ s))\\ &\Rightarrow (\lambda x.x)(\lambda s.(s\ s))\\ &\Rightarrow (\lambda s.(s\ s))\end{aligned}$$

- Reducción de orden aplicativo:

$$\begin{aligned}(\lambda s.(s\ s))(\lambda x.x)(\lambda s.(s\ s)) &\Rightarrow ((\lambda x.x)(\lambda x.x))(\lambda s.(s\ s))\\ &\Rightarrow (\lambda x.x)(\lambda s.(s\ s))\\ &\Rightarrow (\lambda s.(s\ s))\end{aligned}$$

2. (20 puntos) **Programación en Haskell**

Escriba una función recursiva que devuelva la suma desde un valor entero a hasta b.

$$\texttt{sumDesdeHasta a b} \Rightarrow a + (a+1) + (a+2) + \ldots + (b-1) + b$$

Solution:

Una posible solución:

```
1 sumDesdeHasta :: Integer -> Integer -> Integer
2 sumDesdeHasta a b = if a == b
3                     then b
4                     else a + sumDesdeHasta (a+1) b
```

3. (20 puntos) **Lenguaje Calc**
Se ha modificado el lenguaje Calc para que acepte dos nuevos nuevos constructores, el operador unario - (menos unario) y el operador binario + (más unario). Y la siguiente es la nueva gramática $LL(1)$:

$$
\begin{aligned}
Prog &\rightarrow Expr \text{ EOF} \\
Expr &\rightarrow Term\ RestExpr \\
RestExpr &\rightarrow +\ Term\ RestExpr \mid -\ Term\ RestExpr \mid \epsilon \\
Term &\rightarrow Storable\ RestTerm \\
RestTerm &\rightarrow *\ Storable\ RestTerm \mid /\ Storable\ RestTerm \mid \epsilon \\
Storable &\rightarrow UnaryOperators\ (\text{S} \mid \epsilon) \\
UnaryOperators &\rightarrow (- \mid + \mid \epsilon)\ Factor \\
Factor &\rightarrow \text{number} \mid textR \mid '(' \ Expr\ ')'
\end{aligned}
$$

Responda:

a) Indique con precisión: ¿Qué partes de la calculadora deben ser modificadas para aceptar estos nuevos cambios?

b) Implemente la clase o clases que harían parte del nuevo árbol abstracto sintáctico (AST).

Solution:

a) El scanner no cambia. El parser debe cambiar para añadir una nueva producciones llamadas `UnaryOperatoers`. Y se debe añadir dos clases nuevas que representa cada uno de los nuevos operadores unarios: `UnitNeg` y `UnitPos`

b)

```ruby
class UnitNeg
  def initialize(subTree)
    super(subTree)
  end
  def evaluate()
    return (- subTree.evaluate())
  end
end

class UnitPos
  def initialize(subTree)
    super(subTree)
  end
  def evaluate()
    return (subTree.evaluate())
  end
end
```

4. (40 puntos) **Programando en Ruby y en Haskell**

La siguiente es una declaración de una función recursiva que calcula la raíz cuadrada:

$$iraiz(n,a) = \begin{cases} a^2 > n & 0 \\ a^2 \leq n & \textbf{sea } r' = iraiz(n,2a) \textbf{ en } \begin{cases} n < (r'+a)^2 & r' \\ n \leq (r'+a)^2 & r'+a \end{cases} \end{cases}$$

Realice:

a) (20) la implementación en Ruby de la anterior función;

b) (20) la implementación en Haskell de la anterior función.

Nota: Recuerde que en Haskell existe la construcción `let` que permite la asignación de valores a variables (no se pueden modificar posteriormente), por ejemplo:

a) Una posible solución en Ruby:

```ruby
#!/usr/bin/ruby

def iraiz(n,a)
  if a*a > n
    return 0
  else
    rp = iraiz(n,2*a)
    if n < (rp + a)*(rp + a)
      return rp
    else
      return (rp + a)
    end
  end
end
```

b) Una posible solución en Haskell:

```haskell
iraiz :: Integer -> Integer -> Integer
iraiz n a = if a*a > n
            then 0
            else let r'   = iraiz(n,2*a)
                     ra'  = r' + a
                     dra' = ra' * ra'
                 in if n < dra'
                    then r'
                    else ra'
```

ST0244 Lenguajes de programación 031 **Universidad EAFIT**
Supletorio tercer parcial **DIS**
Juan Francisco Cardona McCormick **5 de noviembre de 2013**

Nombre: _____
Código: _____

1. (20 puntos) **Cálculo lambda**

   Evaluar las dos siguiente expresiones λ a su forma normal utilizando los dos mecanismos de evaluación: reducción de orden normal y reducción de orden aplicativo en cada una de las expresiones lambda.

   a) $(\lambda s.(s\ s))(\lambda s.(s\ s))(\lambda s.(s\ s))$

   b) $(\lambda x.x)(\lambda s.(s\ s))(\lambda s.(s\ s))$

2. (20 puntos) **Programación en Haskell**

   Escriba una función recursiva `potencia` que calcula la potencia a^b, donde a y b son valores enteros muy grandes:

   $$\texttt{potencia a b} \Rightarrow \underbrace{a * a * \ldots * a}_{b \text{ veces}}$$

3. (20 puntos) **Lenguaje Calc**
   Se ha modificado el lenguaje Calc para que permita manejar más de una memoria a través de posiciones simbólicas de memoria llamados identificadores `ident`. La siguiente es la nueva gramática $LL(1)$:

   $$\begin{aligned}
   Prog &\rightarrow Expr \text{ EOF} \\
   Expr &\rightarrow Term\ RestExpr \\
   RestExpr &\rightarrow +\ Term\ RestExpr \mid -\ Term\ RestExpr \mid \epsilon \\
   Term &\rightarrow Storable\ RestTerm \\
   RestTerm &\rightarrow *\ Storable\ RestTerm \mid /\ Storable\ RestTerm \mid \epsilon \\
   Storable &\rightarrow Factor \text{ (S ident } \mid \epsilon) \\
   Factor &\rightarrow \text{number} \mid \mid \text{'('}\ Expr\ \text{')'} \mid \text{ident}
   \end{aligned}$$

   Responda:

   a) Indique con precisión: ¿Qué partes de la calculadora deben ser modificadas para aceptar estos nuevos cambios?

   b) Implemente la clase o clases que harían parte del nuevo árbol abstracto sintáctico (AST), o muestre las clases que han sido modificadas.

4. (40 puntos) **Programando en Ruby y en Haskell**

   La siguiente función $iraiz_2$ es otra versión para calcular la raíz cuadrada:

$$iraiz_2(n, a) = \begin{cases} (a+1)^2 > n & a \\ (a+1)^2 \leq n & iraiz_2(n, a+1) \end{cases}$$

   Realice:

   a) (20) la implementación en Ruby de la anterior función;

   b) (20) la implementación en Haskell de la anterior función.

   Nota: Recuerde que en Haskell existe la construcción `let` que permite la asignación de valores a variables (no se pueden modificar posteriormente), por ejemplo:

Nombre: _____

Código: _____

Indicación: Sea breve en sus respuestas y justifiquelas concisamente. Excepto las preguntas de selección múltiple.

1. (20 puntos) **De UML a Ruby**

   En la siguiente figura 1 hay un diagrama de clases diseñado para muchos lenguajes orientados a objeto. En el diagrama hay muchas características que no están definidas directamente en el lenguaje Ruby, por ejemplo: las clases abstractas. En Ruby también existen varias características que simplifican el código generado desde una implementación.

   Implemente el diagrama de clases definido en la figura 1 en el lenguaje Ruby simplificando al máximo el diagrama en el lenguaje Ruby.

   Nota: Recuerde que las clases abstractas no existen en Ruby, el polimorfismo está manejado de forma diferente al lenguaje de programación C++.

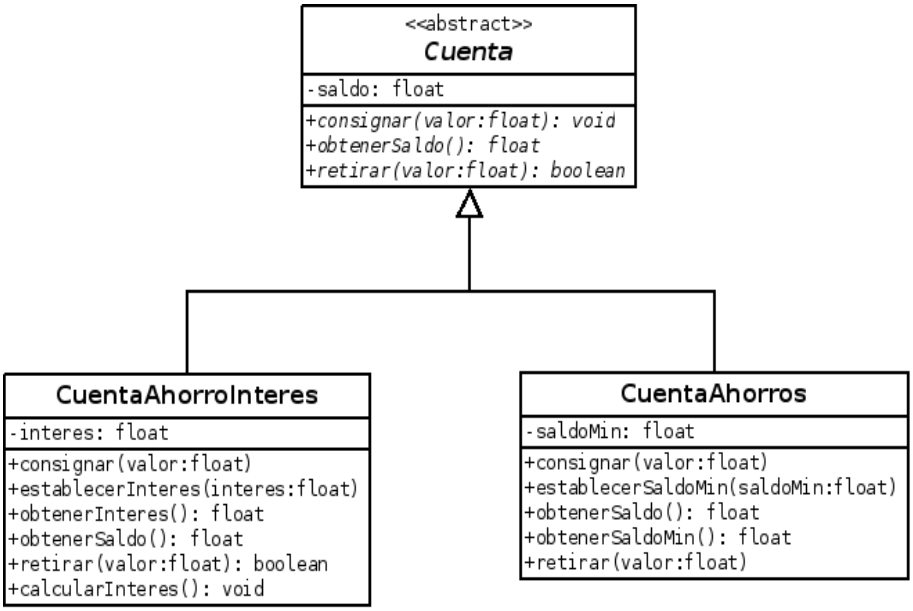


Figura 1: Diagrama de una jerarquía de clases

Solution: Una posible solución:

```ruby
class CuentaAhorros
  attr_reader :saldo
```

```
3   attr_accesso1r :saldoMin
4
5   def initialize(saldo, saldoMin)
6     @saldo = saldo
7     @saldoMin = saldoMin
8   end
9
10  def consignar(valor)
11    @saldo += valor
12  end
13
14  def retirar(valor)
15    if @saldo - valor > saldoMin then
16      @saldo -= saldo
17      return true
18    end
19    return false
20  end
21 end
22
23 class CuentaAhorroInteres
24   attr_reader :saldo
25   attr_accessor :interes
26
27   def initialize(saldo, interes)
28     @saldo = saldo
29     @interes = interes
30   end
31
32   def consignar(valor)
33     @saldo += valor
34   end
35
36   def retirar(valor)
37     if @saldo > valor then
38       @saldo -= valor
39       return true
40     end
41     return false
42   end
43
44   def calcularInteres
45     @saldo = @saldo * @interes
46   end
47 end
```

2. (20 puntos) **Cálculo lambda**

Evaluar las dos siguiente expresiones λ a su forma normal utilizando los dos mecanismos de evaluación: reducción de orden normal y reducción de orden aplicativo en cada una de las expresiones lambda.

a) $(\lambda xy.x)(\lambda x.x)(\lambda y.y)$

b) $(\lambda xy.x)((\lambda x.x)(\lambda y.y))$

Solution:

a)

- Reducción de orden normal:

$$\begin{aligned}(\lambda xy.x)(\lambda x.x)(\lambda y.y) &\Rightarrow (\lambda y.(\lambda x.x))(\lambda y.y)\\ &\Rightarrow (\lambda x.x)\end{aligned}$$

- Reducción de orden aplicativo:

$$\begin{aligned}(\lambda xy.x)(\lambda x.x)(\lambda y.y) &\Rightarrow (\lambda y.(\lambda x.x))(\lambda y.y)\\ &\Rightarrow (\lambda x.x)\end{aligned}$$

b)

- Reducción de orden normal:

$$\begin{aligned}(\lambda xy.x)((\lambda x.x)(\lambda y.y)) &\Rightarrow (\lambda xy.((\lambda x.x)(\lambda y.y)))\\ &\Rightarrow (\lambda y.(\lambda y.y))\end{aligned}$$

- Reducción de orden aplicativo:

$$\begin{aligned}(\lambda xy.x)((\lambda x.x)(\lambda y.y)) &\Rightarrow (\lambda xy.x)(\lambda y.y)\\ &\Rightarrow (\lambda y.(\lambda y.y))\end{aligned}$$

3. (20 puntos) **Programación en Haskell**

Diseñar una función recursiva en Haskell para calcular multiplicaciones según el siguiente método conocido como *del campesino egipcio*:

$$mult(x, y) = \begin{cases} 0 & \text{si } y = 0, \\ x & \text{si } y = 1, \\ mult(2 \times x, y \textbf{ div } 2) & \text{si } y \geq 2 \wedge y \textbf{ mod } 2 = 0, \\ mult(2 \times x, y \textbf{ div } 2) + x & \text{si } y \geq 2 \wedge y \textbf{ mod } 2 = 1, \end{cases}$$

Recuerde que las funciones `div` es la división de enteros en Haskell y `mod` es la función que calcula el residuo.

Solution:

Una posible solución:

```
mult :: Integer -> Integer -> Integer
mult x y = if y == 0
           then 0
           else if y == 1
                then x
                else if y >= 2 && y `mod` 2 == 0
                     then mult (2 * x) (y `div` 2)
                     else mult (2 * x) (y `div` 2) + x
```

Otra posible solución con *pattern matching*:

```
mult :: Integer -> Integer -> Integer
mult x 0 = 0
mult x 1 = x
mult x y
    | y >= 2 && y `mod` 2 == 0 = mult (2 * x) (y `div` 2)
    | y >= 2 && y `mod` 2 == 1 = mult (2 * x) (y `div` 2) + x
```

4. (20 puntos) **Lenguaje Calc**
Se ha modificado el lenguaje Calc para que acepte una nuevo constructor, el operador unario - (menos unario). Y la siguiente es la nueva gramática $LL(1)$:

$$
\begin{aligned}
Prog &\rightarrow Expr \text{ EOF} \\
Expr &\rightarrow Term\ RestExpr \\
RestExpr &\rightarrow +\ Term\ RestExpr \mid -\ Term\ RestExpr \mid \epsilon \\
Term &\rightarrow Storable\ RestTerm \\
RestTerm &\rightarrow *\ Storable\ RestTerm \mid /\ Storable\ RestTerm \mid \epsilon \\
Storable &\rightarrow Negate\ (\text{S} \mid \epsilon) \\
Negate &\rightarrow (-|\epsilon)\ Factor \\
Factor &\rightarrow \text{number} \mid textR \mid '('\ Expr\ ')'
\end{aligned}
$$

Responda:

a) Indique con precisión: ¿Qué partes de la calculadora deben ser modificadas para aceptar este nuevo cambio?

b) Implemente la clase o clases que harían parte del nuevo árbol abstracto sintáctico (AST).

Solution:

a) El scanner no cambia. El parser debe cambiar para añadir una nueva producción llamada `Negate`. Se debe añadir una clase nueva llamada `UnitNeg`.

b)

```
class UnitNeg
  def initialize(subTree)
    super(subTree)
  end
  def evaluate()
    return (- subTree.evaluate())
  end
end
```

5. (20 puntos) **Programación en Ruby**

El siguiente programa no es un programa correcto en el lenguaje de programación Ruby.

```
1  #!
2
3  class MiClase
4      int a
5      bool b
6
7      def init(a, b)
8        @a = a
9        @b = b
10     end
11
12     def a()
13       return a
14     end
15
16     def b()
17       return b
18     end
19
20     def metodo()
21       while not @b
22           if @a > 10
23               @b = True
24           end
25           @a += 1
26       end
27     end
28 end
29
```

Realize la siguiente correcciones:

a) Las instrucciones de la línea 1 a la 10 hay 4 errores, reescriba dichas líneas.

b) Las instrucciones de la linea 12 a la 18 sobran, como se pueden reescribir estas línea en Ruby.

c) El método definido entre las líneas 20 y 27 reescríbalo utilizando lógica negativa.

Solution:

a)

```
1 #!/usr/bin/ruby
2 class MiClase
3   def initialize(a,b)
```

```
    @a = a
    @b = b
  end
```

```
class MiClase
  attr_reader :a, :b
```

b)

```
  def metodo()
    until @b
      @b = True unless @a <= 10
    end
  end
```

Nombre: _____
Código: _____

1. (40 puntos) **De UML a C ++ y Ruby**

   En la siguiente figura 1 hay un diagrama de clases diseñado para el manejo de fletes, carga y otros tipos de equipajes. El objetivo principal de este diseño es establecer los costos de los diferentes tipos de elementos que se pueden transportar. La siguiente es una lista de requisitos obtenida de los diseñadores.

   - Las medidas de peso están dadas en gramos.
   - Los bolsos de mano no tienen ningún costo puesto que son llevados por el pasajero.
   - Las maletas no tendrán costo si son llevadas por el pasajero, en caso contrario será de $1.24[1] por kilo.
   - Las subclases de carga siempre tendrán un sobrecosto debido al nivel de riesgo que ellas represente, la siguiente tabla muestra dicho relación.

Rango riesgo	**Sobrecosto**
$0 \leq gradoDeRiesgo < 0{,}5$	$gradoDeRiesgo * 5{,}00 * peso$
$0{,}5 \leq gradoDeRiesgo \leq 1$	$gradoDeRiesgo * 20{,}00 * peso$

   - Un paquete tiene un costo fijo de $20.01 más un costo por kilo de $10.12.
   - Una caja tiene un costo fijo de $10.10 más un costo por kilo de $5.24.

   a) (20 %) Implemente las clases `Maleta` y `Paquete` en C++.
   b) (20 %) Implemente las clases `Caja` y `BolsoDeMano` en Ruby.

   Nota: Recuerde que las clases abstractas no existen en Ruby, el polimorfismo está manejado de forma diferente al lenguaje de programación C++.

   Solution: Una posible solución:

   a) Implementación de `Maleta` y `Paquete`

```
1 class Maleta : public Equipaje {
2 bool esEquipajeDeMano;
3 public:
4     Maleta(double peso, ID id, long nid, bool equipajeMano);
5     double costo() const;
6 };
7
8 class Paquete : public Carga {
```

[1] Dólares

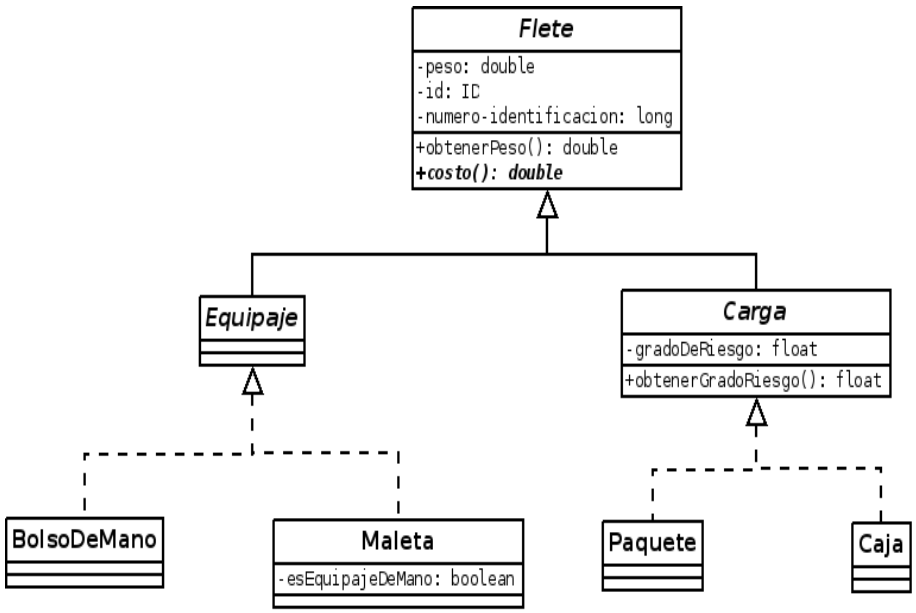


Figura 1: Diagrama de una jerarquía de clases

```
9  public:
10     Paquete(double peso, ID id, long nid, float riesgo);
11     double costo() const;
12 };
13
14 Maleta::Maleta(double peso, ID id, long nid, bool equipajeMano)  :
15   Equipaje(peso, id, nid) , esEquipajeDeMano(equipajeMano) { }
16
17 double Maleta::costo() {
18     if (esEquipajeDeMano) {
19         return 0;
20     }
21     return 1.24 * (obtenerPeso()/1000);
22 }
23
24 Paquete::Paquete(double peso, ID id, long nid, float riesgo) :
25     Carga(peso, id, nid, riesgo) { }
26
27 double Paquete::costo() {
28     double riesgo = obtenerGradoDeRiesgo();
29     double peso = obtenerPeso();
30     double retCosto = 20.01 + 10.12 * (peso / 1000);
31     if (riesgo >= 0.0f and riesgo < 0.5f) {
32         retCosto += riesgo * 5.00 * peso;
33     }
34     else {
35         retCosto += riesgo * 20.00 * peso;
36     }
```

```
37    return retCosto;
38 }
```

b) Implementación de `Caja` y `BolsoDeMano`

```
class Caja < Carga
  def initialize(peso, id, nid, riesgo)
    super(peso, id, nid, riesgo)
  end
  def costo()
    retCosto = 10.10 + 5.24 * (@peso / 1000)
    if @gradoDeRiesgo >= 0 and @gradoDeRiesgo < 0.5 then
      retCosto += @gradoDeRiesgo * 5.00 * @peso
    else
      retCosto += @gradoDeRiesgo * 20.00 * @peso
    end
    return retCosto
  end
end

def BolsoDeMano < Equipaje
  def initialize(peso, id, nid)
    super(peso, id, nid)
  end
  def costo()
    return 0.0;
  end
end
```

2. (25 puntos) **Errores en Ruby**
   El siguiente código en Ruby contiene varios errores. Corriga los errores de modo que el interpretador de Ruby los reconozca como un código válido.

```
class X : Y {
  attr_reader :x
  X(int x) { @x = x }
}
```

Solution:

```
class X < Y
  attr_reader :x
  def initialize(x)
    @x = x
  end
end
```

3. (25 puntos) **Lenguaje Calc**
Se ha modificado el lenguaje Calc para que permita manejar operadores unarios. La siguiente es la nueva gramática $LL(1)$:

$$
\begin{aligned}
Prog &\rightarrow Expr \text{ EOF} \\
Expr &\rightarrow Term\ RestExpr \\
RestExpr &\rightarrow +\ Term\ RestExpr \mid -\ Term\ RestExpr \mid \epsilon \\
Term &\rightarrow Storable\ RestTerm \\
RestTerm &\rightarrow *\ Storable\ RestTerm \mid /\ Storable\ RestTerm \mid \epsilon \\
Storable &\rightarrow Neg\ (\text{S} \mid \epsilon) \\
Neg &\rightarrow (- \mid + \mid \epsilon)\ Factor \\
Factor &\rightarrow \text{number} \mid \text{R} \mid \text{'('}\ Expr\ \text{')'}
\end{aligned}
$$

Responda:

a) Indique con precisión: ¿Qué partes de la calculadora deben ser modificadas para aceptar estos nuevos cambios?

b) Implemente la clase o clases que harían parte del nuevo árbol abstracto sintáctico (AST), o muestre las clases que han sido modificadas.

Solution:

a) En el módulo del `ast` se debe incluir una clase que represente la negación, el valor positivo no es necesario en la semántica actual.

b) Implementación:

```
1  class Neg : public UnaryNode {
2  Neg(AST* subTree);
3  int evaluate() const;
4  };
5
6  Neg::Neg(AST* subTree) : UnaryNode(subTree) { }
7
8  int Neg::evaluate() {
9     return -subTree()->evaluate;
10 }
```

4. (10 puntos) **Ruby**
El siguiente código muestra un implementación de una clase de Ruby de un punto en tres dimensiones:

```ruby
class Punto
  attr_writer :x
  attr_accessor :y
  attr_reader :z

  def initialize(x,y,z)
    @x = x
    @y = y
    @z = z
  end
end
```

Suponga por un momento que Ruby no tiene las palabras reservadas `attr_reader`, `attr_accessor` y `attr_writer`; como debería ser escrita la clase anterior para que cumpla lo que el actual código de Ruby permite.

Solution:

```ruby
class Punto
  def initialize(x,y,z)
    @x = x
    @y = y
    @z = z
  end
  def x=(x)
    @x = x
  end
  def y()
    return @y
  end
  def y=()
    @y = y
  end
  def z()
    return z
  end
end
```

Nombre: _____
Código: _____

1. (25 puntos) **De UML a C++**

   En la siguiente figura 1 hay un diagrama de clases diseñado para muchos lenguajes orientados a objeto.

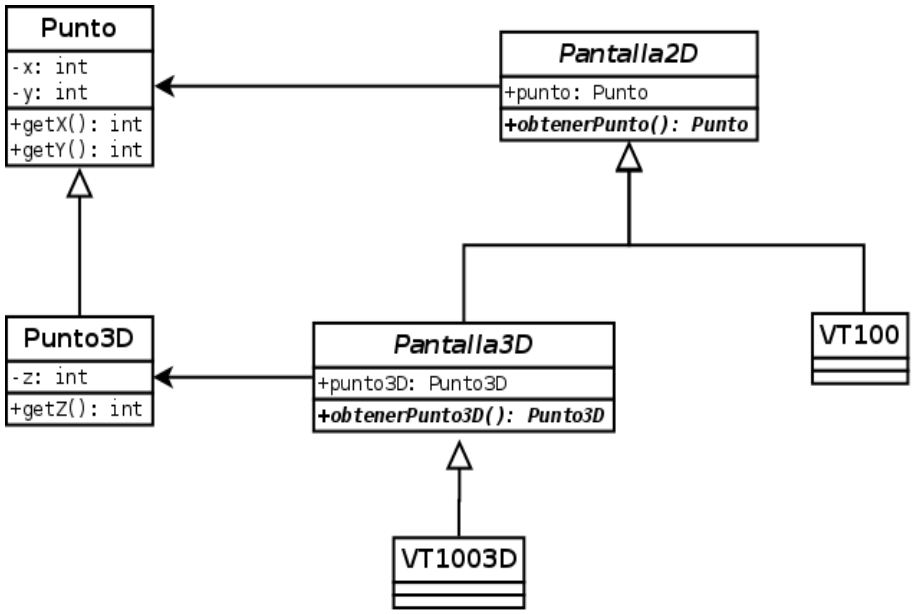


Figura 1: Diagrama de una jerarquía de clases

Implemente las clases `VT100` y `VT1003D` en C++ de tal manera que dichas clases no sean abstractas.

Solution: Una posible solución:

```
class VT100 : public Pantalla2D {
public:
  VT100(Punto p);
  Punto obtenerPunto() const;
};

class VT1003D : public Pantalla3D {
  VT1003D(Punto p, Punto3D p);
  Punto obtenerPunto() const;
  Ponto3D obtenerPunto() const;
};

VT100::VT100(Punto p) : punto(p) { }

```

```
15 Punto VT100::obtenerPunto() {
16     return punto;
17 }
18
19 VT1003D::VT1003D(Punto p, Punto3D p3) : punto(p), punto3D(p3) { }
20
21 Punto VT1003D::obtenerPunto() {
22     return punto;
23 }
24
25 Punto3D VT1003D::obtenerPunto3D() {
26     return punto3D;
27 }
```

2. (25 puntos) **Errores en C++**
   El siguiente código contiene cuatro errores en C++ podría corregirlos para que la clase compile correctamente:

```
class X : nopublic Y {
private
  int x;
public:
  X();
  int getX();
}
```

Solution:

```
class X : private Y {
private:
  int x;
public:
  X();
  int getX() const;
};
```

3. (25 puntos) **Lenguaje Calc**
Se ha modificado el lenguaje Calc para que permita manejar más de una memoria a través de identificación por valores enteros `number`. La siguiente es la nueva gramática $LL(1)$:

$$
\begin{aligned}
Prog &\rightarrow Expr \text{ EOF} \\
Expr &\rightarrow Term\ RestExpr \\
RestExpr &\rightarrow +\ Term\ RestExpr \mid -\ Term\ RestExpr \mid \epsilon \\
Term &\rightarrow Storable\ RestTerm \\
RestTerm &\rightarrow *\ Storable\ RestTerm \mid /\ Storable\ RestTerm \mid \epsilon \\
Storable &\rightarrow Factor \text{ (S number } \mid \epsilon) \\
Factor &\rightarrow \text{number} \mid \text{(R number)} \mid \text{'('}\ Expr\ \text{')'}
\end{aligned}
$$

Responda:

a) Indique con precisión: ¿Qué partes de la calculadora deben ser modificadas para aceptar estos nuevos cambios?

b) Implemente la clase o clases que harían parte del nuevo árbol abstracto sintáctico (AST), o muestre las clases que han sido modificadas.

Solution:

a) El parser `parser.C` debe ser modificado en el método `Storable` para introduccir el manejo del número en el constructor `S`. También dentro del parser el método `Factor` para el manejo del número en el constructor `R`.
En el árbol abstracto sintáctico (`AST`), se modificará el `ast.h` y `ast.C` para modificar las clases `StoreNode` y `RecalNode` para incluir ahora un número.
También la clase `Calculator` para incluir un arreglo de memorias, y modificar los métodos `store` y `recall`.

b) 1) Clase `StoreNode`

```
class StoreNode : public UnaryNode {
  int memory;
 public:
  StoreNode(AST* sub, int mem);

  int evaluate();
};

StoreNode::StoreNode(AST* sub, int mem) :
  UnaryNode(sub), memory(mem) { }


int StoreNode::evaluate() {
  int value = getSubTree()->evaluate();
  calc->store(value,memory);
```

```
16    return value;
17  }
```

2) Clase `RecallNode`

```
1  class RecallNode : public AST {
2    int memory;
3   public:
4    RecallNode(int mem);
5
6    int evaluate();
7  };
8
9  RecallNode::RecallNode(int mem)
10   : AST(), memory(mem) { }
11
12 int RecallNode::evaluate() {
13   return calc->recall(memory);
14 }
```

4. (25 puntos) **Ruby**
   El siguiente código muestra un implementación de una clase de Ruby de la conocida clase Punto:

```ruby
class Punto
  attr_reader :x, :y

  def initialize(x,y)
    @x = x
    @y = y
  end
end
```

Suponga por un momento que Ruby no tiene la palabra reservada `attr_reader` como debería ser escrita la clase anterior para que cumpla lo que el actual código de Ruby permite.

Solution:

```ruby
class Punto
  def initialize(x,y)
    @x = x
    @y = y
  end
  def x()
    return @x
  end
  def y()
    return @y
  en
end
```

Nombre: _____

Código: _____

Indicación: Sea breve en sus respuestas y justifiquelas concisamente. Excepto las preguntas de selección múltiple.

1. (25 points) **Gramáticas independientes de contexto**

En la figura 1 se muestra una gramática de un lenguaje de bloques $G_{\text{bloques}} = (V, \Sigma, P, S)$.

$$
\begin{array}{lcl}
V & = & \{B, S, A, I\} \\
\Sigma & = & \{\ '[',\ ']',\ a,\ ';'\ \} \\
P & = & \{B \rightarrow\ '['\ S\ ']'\ \mid\ \epsilon, \\
 & & S\ \rightarrow\ \epsilon\ \mid\ A, \\
 & & A \rightarrow\ I\ ';'\ A\ \mid\ I, \\
 & & I \rightarrow a\ \mid\ B, \\
S & = & B
\end{array}
$$

Figura 1: Gramática de bloques

a) Muestre la derivación más a la izquierda que produzca la cadena siguiente cadena [[[]]].

b) Muestre la derivación más a la derecha que produzca la cadena [a; [[]]; []].

Solution:

a) Derivación más a la izquierda:

$$
\begin{array}{rcl}
B & \Rightarrow & [S\,] \\
 & \Rightarrow & [A] \\
 & \Rightarrow & [I] \\
 & \Rightarrow & [B] \\
 & \Rightarrow & [[S\,]] \\
 & \Rightarrow & [[A]] \\
 & \Rightarrow & [[I]] \\
 & \Rightarrow & [[B]] \\
 & \Rightarrow & [[[S\,]]] \\
 & \Rightarrow & [[[\epsilon]]]
\end{array}
$$

b) Derivación más a la derecha:

$$
\begin{aligned}
B &\Rightarrow [S\,] \\
&\Rightarrow [A] \\
&\Rightarrow [I\ ;\ A] \\
&\Rightarrow [I\ ;\ I\ ;\ A] \\
&\Rightarrow [I\ ;\ I\ ;\ B] \\
&\Rightarrow [I\ ;\ I\ ;\ [S\,]] \\
&\Rightarrow [I\ ;\ I\ ;\ [\epsilon]] \\
&\Rightarrow [I\ ;\ B\ ;\ []] \\
&\Rightarrow [I\ ;\ [S\,]\ ;\ []] \\
&\Rightarrow [I\ ;\ [A]\ ;\ []] \\
&\Rightarrow [I\ ;\ [I]\ ;\ []] \\
&\Rightarrow [I\ ;\ [B]\ ;\ []] \\
&\Rightarrow [I\ ;\ [[S\,]]\ ;\ []] \\
&\Rightarrow [I\ ;\ [[\epsilon]]\ ;\ []] \\
&\Rightarrow [a\ ;\ [[]]\ ;\ []]
\end{aligned}
$$

2. (25 points) **EWE**
   Escriba un programa en EWE que lea tres números enteros y escriba el menor de los tres.

Solution: Una posible solución:

```
main: readInt(a)
      readInt(b)
      readInt(c)
      if a >= b then goto l1
      if a >= c then goto l2
      writeInt(a)
      goto end
l1:   if b >= c then goto l2
      writeInt(b)
      goto end
l2:   writeInt(c)
end:  halt
equ a M[0]
equ b M[1]
equ c M[2]
```

3. (25 points) **Gramáticas**
Considere nuevamente la gramática de la figura 1, la siguiente cadena es una cadena válida de dicha gramática:

$$[\ a\ ;\ [\ a\ ;\ a\ ;\ a\ ]\ ;\ a\ ]$$

a) Construya el árbol de derivación correspondiente.

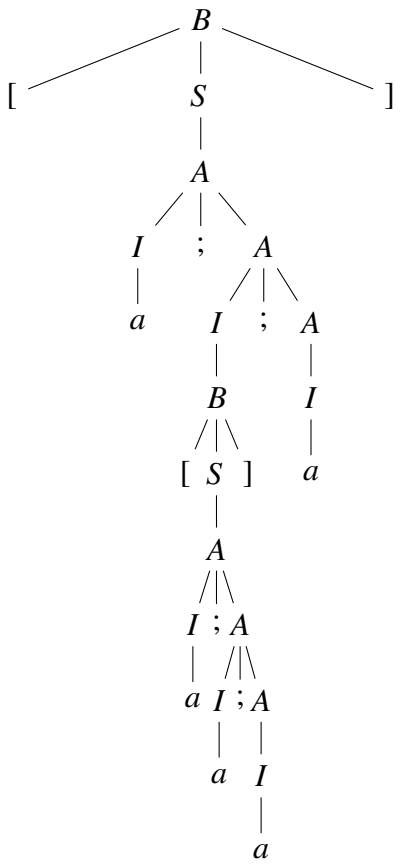
b) Del árbol obtenido en el punto anterior construir el correspondiente árbol abstracto.

Solution:

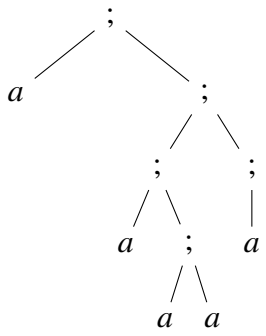
1. En primer lugar se construye la derivación:

$$\begin{aligned}
B &\Rightarrow [S\,] \\
&\Rightarrow [A] \\
&\Rightarrow [I\ ;\ A] \\
&\Rightarrow [a\ ;\ A] \\
&\Rightarrow [a\ ;\ I\ ;\ A] \\
&\Rightarrow [a\ ;\ B\ ;\ A] \\
&\Rightarrow [a\ ;\ [S\,]\ ;\ A] \\
&\Rightarrow [a\ ;\ [A]\ ;\ A] \\
&\Rightarrow [a\ ;\ [I\ ;\ A]\ ;\ A] \\
&\Rightarrow [a\ ;\ [a\ ;\ A]\ ;\ A] \\
&\Rightarrow [a\ ;\ [a\ ;\ I\ ;\ A]\ ;\ A] \\
&\Rightarrow [a\ ;\ [a\ ;\ a\ ;\ A]\ ;\ A] \\
&\Rightarrow [a\ ;\ [a\ ;\ a\ ;\ I]\ ;\ A] \\
&\Rightarrow [a\ ;\ [a\ ;\ a\ ;\ a]\ ;\ A] \\
&\Rightarrow [a\ ;\ [a\ ;\ a\ ;\ a]\ ;\ I] \\
&\Rightarrow [a\ ;\ [a\ ;\ a\ ;\ a]\ ;\ a]
\end{aligned}$$

A partir de la derivación, se construye el árbol



2. Con la solución del punto anterior se puede construir el árbol abstracto:



4. (25 points) **Otras gramáticas**

   Como se explicó en clase, son muchos los tipos de gramáticas que existen para mostrar lenguajes de programación, herramientas de cómputo y sistemas de comandos. La siguiente es una sintaxis para un lenguaje de comandos y el extracto de su manual. Es una combinación de CBL, BNF y otros.

```
(read | write) (tape | disk) [[ rec id] ...| [struct id] | - ]
    on (read | write) (tape | disk) [[ rec id] ... | [struct id] | - ]
```

   Que este comando permite leer o escribir una serie de registros o estructuras o cualquier cosa de una cinta o disco a los correspondientes registros, estructuras o cualquier cosa en disco o cinta. Los registros y estructuras están identificados por `id` que es un nombre de variable.

   Muestre:

   a) Como utilizar la anterior gramática para leer dos registros *a* y *b* de cinta y escribirlos en disco los correspondientes registros.

   b) Esta gramática permite que se hagan ciertas operaciones que no son válidas, muestre un ejemplo del uso de la gramática de forma que sintácticamente es válido pero semánticamente no tiene sentido.

Solution:

a) `read tape rec a rec b on write disk rec a rec b`

b) La siguiente es una lisa no exhaustiva de operaciones válidas desde el punto de vista gramátical pero invalidas desde el punto de vista lógico:

   1- Leer un registro y escribir uno que no se leyó:

   ```
   read tape rec a on write disk rec b
   ```

   2- leer una estructura y escribir un registro que no se leyó:

   ```
   read tape struct a on write disk rec b
   ```

   3- Leer un registro y leer en otro registro:

   ```
   read tape rec a on read tape b
   ```

Nombre: _____

Código: _____

Indicación: Sea breve en sus respuestas y justifiquelas concisamente. Excepto las preguntas de selección múltiple.

1. (25 points) **Gramáticas independientes de contexto**

En la figura 1 se muestra una gramática de un lenguaje de bloques $G_{\text{bloques}} = (V, \Sigma, P, S)$.

$$
\begin{array}{rcl}
V & = & \{B, S, A, I\} \\
\Sigma & = & \{\ '[',\ ']',\ a,\ ';'\ \} \\
P & = & \{B \rightarrow\ '['\ S\ ']'\ \mid\ \epsilon, \\
 & & S\ \rightarrow\ \epsilon\ \mid\ A, \\
 & & A \rightarrow\ A\ ';'\ I\ \mid\ I, \\
 & & I \rightarrow a\ \mid\ B, \\
S & = & B
\end{array}
$$

Figura 1: Gramática de bloques

a) Muestre la derivación más a la izquierda que produzca la cadena siguiente cadena [[[]]].

b) Muestre la derivación más a la derecha que produzca la cadena [a; [[]]; []].

Solution:

a) Derivación más a la izquierda:

$$
\begin{array}{rcl}
B & \Rightarrow & [S\,] \\
 & \Rightarrow & [A] \\
 & \Rightarrow & [I] \\
 & \Rightarrow & [B] \\
 & \Rightarrow & [[S\,]] \\
 & \Rightarrow & [[A]] \\
 & \Rightarrow & [[I]] \\
 & \Rightarrow & [[B]] \\
 & \Rightarrow & [[[S\,]]] \\
 & \Rightarrow & [[[\epsilon]]]
\end{array}
$$

b) Derivación más a la derecha:

$$
\begin{aligned}
B &\Rightarrow [S\,] \\
&\Rightarrow [A] \\
&\Rightarrow [A\ ;\ I] \\
&\Rightarrow [A\ ;\ B] \\
&\Rightarrow [A\ ;\ [S\,]] \\
&\Rightarrow [A\ ;\ [\epsilon]] \\
&\Rightarrow [A\ ;\ I\ ;\ [\epsilon]] \\
&\Rightarrow [A\ ;\ B\ ;\ []] \\
&\Rightarrow [A\ ;\ [S\,]\ ;\ []] \\
&\Rightarrow [A\ ;\ [A]\ ;\ []] \\
&\Rightarrow [A\ ;\ [I]\ ;\ []] \\
&\Rightarrow [A\ ;\ [B]\ ;\ []] \\
&\Rightarrow [A\ ;\ [[]]\ ;\ []] \\
&\Rightarrow [I\ ;\ [[]]\ ;\ []] \\
&\Rightarrow [a\ ;\ [[]]\ ;\ []]
\end{aligned}
$$

2. (25 points) **EWE**
   Escriba un programa en EWE que lea tres números enteros y escriba el mayor de los tres.

Solution: Una posible solución:

```
main: readInt(a)
      readInt(b)
      readInt(c)
      if a <= b then goto l1
      if a <= c then goto l2
      writeInt(a)
      goto end
l1:   if b <= c then goto l2
      writeInt(b)
      goto end
l2:   writeInt(c)
end:  halt
equ   a M[0]
equ   b M[1]
equ   c M[2]
```

3. (25 points) **Gramáticas**
Considere nuevamente la gramática de la figura 1, la siguiente cadena es una cadena válida de dicha gramática:

$$[\ a\ ;\ [\ a\ ;\ a\ ;\ a\ ]\ ;\ a\ ]$$

a) Construya el árbol de derivación correspondiente.

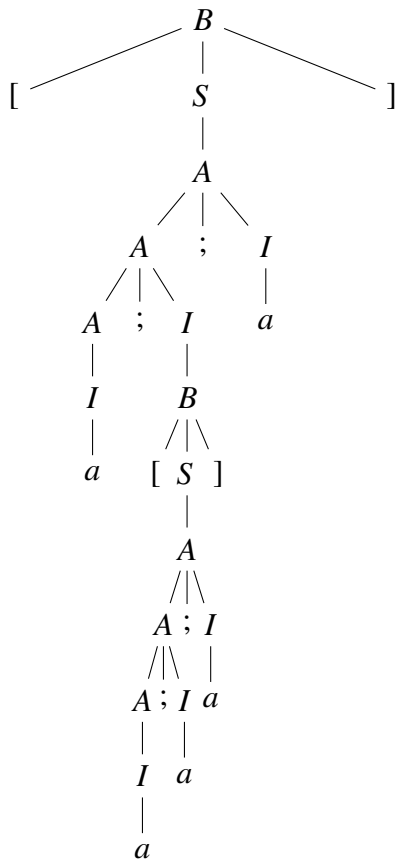
b) Del árbol obtenido en el punto anterior construir el correspondiente árbol abstracto.

Solution:

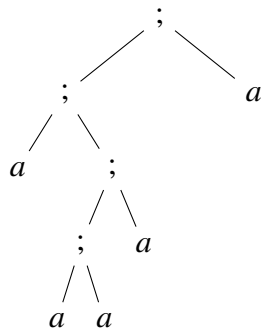
a) En primer lugar se construye la derivación:

$$\begin{aligned}
B &\Rightarrow [S] \\
&\Rightarrow [A] \\
&\Rightarrow [A\ ;\ I] \\
&\Rightarrow [A\ ;\ I\ ;\ I] \\
&\Rightarrow [I\ ;\ I\ ;\ I] \\
&\Rightarrow [a\ ;\ I\ ;\ I] \\
&\Rightarrow [a\ ;\ B\ ;\ I] \\
&\Rightarrow [a\ ;\ [S]\ ;\ I] \\
&\Rightarrow [a\ ;\ [A]\ ;\ I] \\
&\Rightarrow [a\ ;\ [A\ ;\ I]\ ;\ I] \\
&\Rightarrow [a\ ;\ [A\ ;\ I\ ;\ I]\ ;\ I] \\
&\Rightarrow [a\ ;\ [I\ ;\ I\ ;\ I]\ ;\ I] \\
&\Rightarrow [a\ ;\ [a\ ;\ I\ ;\ I]\ ;\ I] \\
&\Rightarrow [a\ ;\ [a\ ;\ a\ ;\ I]\ ;\ I] \\
&\Rightarrow [a\ ;\ [a\ ;\ a\ ;\ a]\ ;\ I] \\
&\Rightarrow [a\ ;\ [a\ ;\ a\ ;\ a]\ ;\ a]
\end{aligned}$$

A partir de la derivación, se construye el árbol:



b) Con la solución del punto anterior se puede construir el árbol abstracto:



4. (25 points) **Otras gramáticas**

   Como se explicó en clase, son muchos los tipos de gramáticas que existen para mostrar lenguajes de programación, herramientas de cómputo y sistemas de comandos. La siguiente es una sintaxis para un lenguaje de comandos y el extracto de su manual. Es una combinación de CBL, BNF y otros.

```
(read | write) (tape | disk) [[ rec id] ...| [struct id] | - ]
    on (read | write) (tape | disk) [[ rec id] ... | [struct id] | - ]
```

   Que este comando permite leer o escribir una serie de registros o estructuras o cualquier cosa de una cinta o disco a los correspondientes registros, estructuras o cualquier cosa en disco o cinta. Los registros y estructuras están identificados por `id` que es un nombre de variable.

   Muestre:

   a) Como utilizar la anterior gramática para leer dos estructuras *a* y *b* de disco y escribirlos a cinta los correspondientes registros.

   b) Esta gramática permite que se hagan ciertas operaciones que no son válidas, muestre un ejemplo del uso de la gramática de forma que sintácticamente es válido pero semánticamente no tiene sentido.

Solution:

a) `read tape struct a struct b on write disk struct a struct b`

b) La siguiente es una lisa no exhaustiva de operaciones válidas desde el punto de vista gramátical pero invalidas desde el punto de vista lógico:

   1- Leer un registro y escribir uno que no se leyó:

   ```
   read tape rec a on write disk rec b
   ```

   2- leer una estructura y escribir un registro que no se leyó:

   ```
   read tape struct a on write disk rec b
   ```

   3- Leer un registro y leer en otro registro:

   ```
   read tape rec a on read tape b
   ```

ST0244 Lenguajes de programación 031 **Universidad EAFIT**
Segundo parcial **DIS**
Juan Francisco Cardona McCormick **4 de octubre de 2014**

Nombre: _____
Código: _____

1. (25 points) **Cadenas de carácteres**
   Escriba una función `saltarblancosinicio` que se encarga de reemplazar los espacios en blanco al inicio de una cadena de carácteres por el carácter `'\0'`. A continuación su propotipo:

```
1 char* borrarblancosinicio(char *s);
```

   Esta función retorna la dirección del primer carácter no blanco.

   Solution: Una posible solución:

```
1 char* borrarblancosinicio(char *s) {
2   while (s  and *s == ' ' and *s) {
3     *s = '\0';
4     s++;
5   }
6   return s;
7 }
```

2. (25 points) **Tipos de datos en C++ y argumentos de funciones**

   El procedimiento `menosModa`, toma dos argumentos: un apuntador a un arreglo de valores de tipo `entero`[1] y el número total de elementos; calcula la moda[2] de los valores; luego resta a cada valor el valor de la moda encontrado. La función debe modificar los valores pasados. El siguiente código es la signatura (*prototipo* o *firma*) de la función.

   Tenga presente que el procedimiento debe modificar los valores entregados.

```
1 void menosModa(const int* arreglo, const int longitud);
```

   a) ¿Son correctos los parámetros del procedimiento? En caso contrario: modifique los argumentos para que sea adecuados para implementar el procedimiento..

   b) Implemente el procedimiento `menosModa`.

Solution:

```
1  void menosModa(int *arreglo, const int longitud) {
2    int frec[10] = { 0 };
3    int mayor = 0;
4    int indice = 0;
5    for (int i = 0; i < longitud; i++) {
6      frec[arreglo[i]]++;
7      if (frec[arreglo[i]] > mayor) {
8        mayor = frec[arreglo[i]];
9        indice = arreglo[i];
10     }
11   }
12   for (int i = 0;i < longitud; i++) {
13     arreglo[i] -= indice;
14   }
15   return ;
16 }
```

[1] Todos los valores son de un digito

[2] La moda es el valor que tiene mayor frecuencia absoluta

3. (25 points) **Ámbito**
   ¿Cuál es la salida del siguiente programa en C++?

```cpp
#include <iostream>
using namespace std;

int a;

namespace W {
  int a;
  int f() { return a++; }
}

int f() { return ++a; }
int g(int a) { return a + f() + W::f(); }

int
main() {
  int a = W::a + ::a + 1;
  cout << g(a) << endl;
  return 0;
}
```

Solution: El primer paso es la inicialización de la variables globales: `::a = 0` y `W::a = 0`. Luego inicia la función `main` y allí inicia la variable local `a = W::a + ::a + 1` lo que obtenemos es `a = 1`; luego se invoca la función local `g`. En `g` la evaluación toma un valor local `a = 1`, más la invocación de `f` global; esta función obtiene 0 por que aunque incrementa `::a` en uno utiliza una función de incremento prefijo; mientras que al invocar a `W::f`, se incrementa a `W::a = 1` con una función de incremento prefijo, lo que retorna 1; entonces, al sumar 1 + 0 + 1, la función retorna 2.

4. (25 points) **Valores estáticos**

La siguiente función `sc` es un ejemplo que demuestra el uso del modificador `static` en el lenguaje de programación C++.

```
int* sc(int* p, int l, int d) {
  static int *a;
  static int al;

  if (p != NULL and l != -1) {
    al = l;
    a = p;
  }

  while (al > 0 and *a == d) {
    al--;
    a++;
  }

  if (al == 0) return NULL;

  al--;
  return a++;
}
```

La función `sc` es utilizada de una manera muy extraña. La primera vez que es invocada, los argumentos los dos primeros argumentos tienen la dirección válida de un arreglo de enteros y su longitud, el tercer argumento muestra un valor que es utilizado; las siguientes invocaciones los dos primeros argumentos deben ser `NULL` y una longitud negativa.

Explique que hace la función `sc`.

Solution: El siguiente código muestra el ejemplo de utilizar esta función y el resultado.

```
int
main() {

  int arreglo[] = { 0, 10, 20, 11, 0, 21, 34, 33};

  int *p = sc(arreglo, sizeof(arreglo)/sizeof(int),0);
  while (p != NULL) {
    cout << *p << endl;
    p = sc(NULL, -1, 0);
  }
  return 0;
}
```

El resultado de evaluar este programa es la siguiente salida:

```
$ ./punto4
10
20
11
21
34
33
```

La función utiliza recorrera un arreglo eliminando los elementos indicados por la variable `d`.

ST0244 Lenguajes de programación 031 **Universidad EAFIT**
Tercer parcial **DIS**
Juan Francisco Cardona McCormick **8 de noviembre de 2014**

Nombre: _____
Código: _____

1. (25 puntos) **Eliminando el viejo `goto`**
   En una carta al editor de CACM, Rubin[1987] utiliza el siguiente segmento de código como evidencia que la legibilidad de algunos códigos es mejor que el que el equivalente sin `goto`. Este código se encuentra la primera fila de una matriz $n \times n$ con nombre x que tiene únicamente valores cero.

```
1     int i, j;
2     for (i = 0; i < n; i++) {
3        for (j = 0; j < n; j++) {
4           if (x[i][j] != 0)  goto reject;
5        }
6        std::cout << "Primera fila con todo en ceros es: "
7                  << i << std::endl;
8        goto end;
9     reject:
10    }
11    end:
12
```

   La instrucción `goto` requiere de una etiqueta para determinar donde debe continuar su ejecución. En el ejemplo anterior la instrucción `goto reject` salta a la dirección indicada por la etiqueta `reject:` en la línea 9.

   Reescriba correctamente el código anterior en C++, en el cual el nuevo código no pierda la semántica original del programa y no tenga ninguna instrucción `goto`.

   Solution: Una posible solución:

```
1     int i, j;
2     bool found = true;
3     for (i = 0; i < n; i++) {
4        for (j = 0; j < n; j++) {
5           if (x[i][j] != 0)  {
6              found = false;
7              break;
8           }
9        }
10       if (found) {
11          std::cout << "Primera fila con todo en ceros es: "
12                    << i << std::endl;
13          break;
14       }
15       found = true;
16    }
```

2. (25 puntos) **Ciclos en C++**
Los ciclos en C++ son a menudo escritos con cuerpos vacíos colocando todos los efectos colaterales en las pruebas, como en los dos ejemplos siguientes:

```
1    i = 0;
2    while (a[i++] != 0);
3
4    for (i = 0; a[i] != 0; i++);
5
```

a) ¿Son estos ciclos equivalentes? Justifique su respuesta.

b) ¿Existe alguna ventaja o desventaja al utilizar este estilo de código?

Solution:

a) Ambos códigos lucen similares, recorren un arreglo hasta que encuentran un valor 0 en un posición. Donde la equivalencia se rompe es en el valor final de la variable `i`, puesto que en el primer ciclo la variable quedará en una posición más que el segundo ciclo. Si esto es para encontrar el índice donde encontrar el valor cero, en el primer ciclo se debe ajustar el valor en una posición menos, mientras que en el segundo tendríamos el valor correcto. Por lo tanto ambos no son similares.

b)
- Algunas ventajas:
  - Código más compacto, en casi una línea se puede recorrer un arreglo.
  - A ser el código más compacto, genera programas más cortos.
- Algunas desventajas:
  - El código es más díficil de entender. Suponga que se tiene un arreglo de carácteres, el código sirve para encontrar la última posición en este arreglo. Pero si tiene un arreglo de enteros, esta buscando la primera posición de un valor cero.
  - Un programado se confundo por que no encuentra inmediatamente el cuerpo de los ciclos y puede confundir que las siguiente instrucción puede hacer parte del ciclo (si no se cuenta con un buen IDE).

3. (25 puntos) `Evaluación de condiciones en C++`
   Considere la siguiente expresión en C++:

```
1 a/b > 0 and b/a >0
```

   a) ¿Cuál será el resultado de evaluar la expresión cuando a es 0?

   b) ¿Cuál será el resultado de evaluar la expresión cuando b es 0?

Solution:

a) Cuándo el valor de a es cero, inmediatamente en la primera parte de expresión el resultado de a/b es cero por lo tanto por evaluación de corto-circuito la segunda parte no se hace.

b) Cuándo el valor de b es cero, la primera parte de la expresión genera una división por cero, lo que hace que el programa termine inmediatamente.

4. (25 puntos) **Clase en C++**

Dado el siguiente código en C++:

```
1  class A { public:
2        virtual void p() { std::cout << "A.p" << std::endl; }
3        void q() { std::cout << "A.q" << std::endl; }
4        virtual void r() { p(); q(); }
5  };
6
7  class B : public A { public:
8        virtual void p() { std::cout << "B.p" << std::endl; }
9  };
10
11 class C : public B { public:
12     void q() { std::cout << "C.q" << std::endl; }
13     void r() { q(); p(); }
14 };
15
16 int main() {
17   ...
18    A a;
19    C c;
20    a = c;
21    a.r();
22    A* ap = new B;
23    ap->r();
24    ap = new C;
25    ap->r();
26   ...
27 }
```

¿Qué es lo que imprime el código? ¿Por qué?

Solution: Vamos paso a paso. En la línea 18, se crea una instancia de la clase `A`. En la línea 19, se crea un instancia de la clase `C`. En la línea 20, se hace una asignación de la instancia de `C` a una instancia de `A`, esto solamente copia los atributos que son comunes en la clase `A`, la variable `a` sigue siendo una instancia de la clase `A`. En la línea 21, se llama al método `r()` de la clase `A`, por lo tanto invoca primero al método `p()` de la clase `A` y luego al método `q()` de la clase `A`. La salida queda:

```
A.p
A.q
```

En la línea 22, se declara una variable `ap` de tipo apuntador a `A`, pero se inicializa con una dirección de tipo `B`. En la línea 23, se llama al método `r()` en un instancia de clase `B`, pero esta no tiene definido este método así que busca en la clase padre `A`, esta invoca los métodos `p()`, en la clase `B` y el método `q()`, que no se encuentra definido en la clase **B**, así que llama al método en la clase padre `A`, así que la salida queda así:

```
B.p
A.q
```

En la línea 24, se asigna a la variable `ap` una dirección de un tipo **C**, el objeto que apunta es de tipo B aunque este asignado a una variable de tipo dirección de `A`. En la línea 25, se llama al método `r()`, pero como esta clase es de tipo **C** esta llama al método sobrecargado en la clase **C**, que invoca primero al método `q()` y luego a `p()`, pero como en la clase `C` solamente esta sobrecargado `q()`, se llama primero a este método y luego a `p()` en la clase padre que primer lo tenga sobrecargado esto es en la clase `B`, así que la salida queda así:

```
C.q
B.q
```

Resumiendo la salida queda así:

```
A.p
A.q
B.p
A.q
C.q
B.q
```

5. (25 puntos) **`switch-case` en C++**

   A continuación presentamos un enunciado `switch-case` correcto en C++:

```
1 int n = ...;
2 int q = (n + 3) / 4;
3 switch (n %4)
4 { case 0: do { n++;
5   case 3:    n++;
6   case 2:    n++;
7   case 1:    n++;
8   } while (--q > 0);
9 }
```

   Suponga que n tiene el valor de 1. ¿Cuál es el valor de n en cada cada caso, después de la ejecución del `switch-case` anterior.

Solution:

a) Ejecución:
   En la línea 2, se calcula el valor de `q` este es igual a:

$$(n + 3)/4 \equiv (1 + 3)/4 \equiv 4/4 \equiv 1$$

   En la línea 3, se evalúa la expresión de `switch-case` y se obtiene:

$$n\,\%4 \equiv 1\,\%4 \equiv 1$$

   Así que se va por la etiqueta `case 1:` y ejecuta la instrucción `n++`, y obtiene en `n` un valor de 2. En la línea siguiente, línea 8, computa a `q` y obtiene un valor de cero, por lo tanto el ciclo termina y `n` al final queda en 2.

b) ¿Por qué el anterior código es correcto?
   Contrario a lo que la intuición pueda suponer, el compilador trata cada `case` como una etiqueta, al igual que el código del `do` es también otra etiqueta, por lo tanto no hay problema al combinar así la instrucción `do-while` dentro de una instrucción `switch-case`.

ST0244 Lenguajes de programación 031 **Universidad EAFIT**
Supletorioo segundo parcial **DIS**
Juan Francisco Cardona McCormick **27 de octubre de 2014**

Nombre: _____
Código: _____

1. (33 points) **Cadenas de carácteres**
   Escribala función `indiceAlaDerecha` que se encarga de de localizar el último carácter que coincide con `c` (convertido a carácter) en una cadena de carácteres terminado con nulo (`'\0'`).

   El siguiente es el prototipo de la función:

```
const char* indiceAlaDerecha(const char *s, int c);
```

   Esta función retorna la dirección del último carácter `c` en la cadena `s`, si el carácter no existe en la cadena retorna nulo (`'\0'`).

2. (33 points) **Tipos de datos en C++ y argumentos de funciones**

   El procedimiento `cambios`, toma dos argumentos: un apuntador a un arreglo de valores de tipo `float` y el número total de elementos; el programa busca el menor de todos los elementos; luego recorre de nuevo el arreglo retornando en cada posición: 0 si es igual al menor, 1 si es mayor al menor. La función debe modificar los valores pasados. El siguiente código es la signatura (*prototipo* o *firma*) de la función.

   Tenga presente que el procedimiento debe modificar los valores entregados.

```
void cambios(const double* arreglo, const int longitud);
```

   a) ¿Son correctos los parámetros del procedimiento? En caso contrario: modifique los argumentos para que sea adecuados para implementar el procedimiento..

   b) Implemente el procedimiento `cambios`.

3. (34 points) **Ámbito**
   ¿Cuál es la salida del siguiente programa en C++?

```
#include <iostream>

int a;

namespace W { int a;
  int f() { return a++; }
}

int f() { return a++; }
int g(int a) { return (a++) + f() + W::f(); }

int
main() {
  int a = ++W::a + ::a++ + 1;
  std::cout << g(a) << std::endl;
  return 0;
}
```

ST0244 Lenguajes de programación 032 **Universidad EAFIT**
Segundo parcial **DIS**
Juan Francisco Cardona McCormick **4 de octubre de 2014**

Nombre: _____
Código: _____

1. (25 points) **Cadenas de carácteres**
   Escriba una función `saltarblancosfinal` que se encarga de reemplazar los espacios en blanco al final de una cadena de carácteres por el carácter `'\0'`. A continuación su propotipo:

```
char* borrarblancosfinal(char *s);
```

   Esta función retorna la dirección del primer carácter no blanco.

2. (25 points) **Tipos de datos en C++ y argumentos de funciones**

   El procedimiento `medios`, toma dos argumentos: un apuntador a un arreglo de valores de tipo `double` y el número total de elementos; calcula el promedio[1] de todos los valores; luego resta a cada valor el promedio encontrado. La función debe modificar los valores pasados. El siguiente código es la signatura (*prototipo* o *firma*) de la función.

   Tenga presente que el procedimiento debe modificar los valores entregados.

```
void medios(const double* arreglo, const int longitud);
```

   a) ¿Son correctos los parámetros del procedimiento? En caso contrario: modifique los argumentos para que sea adecuados para implementar el procedimiento..

   b) Implemente el procedimiento `medios`.

3. (25 points) **Ámbito**
   ¿Cuál es la salida del siguiente programa en C++?

```
#include <iostream>
using namespace std;

int a;

namespace W {
  int a;
  int f() { return ++a; }
}

int f() { return ++a; }
int g(int a) { return a + f() + W::f(); }

int
main() {
  int a = W::a + ::a + 1;
  cout << g(a) << endl;
  return 0;
}
```

[1] Un promedio es igual a la suma de todos los elementos dividido el número total de elementos

4. (25 points) **Valores estáticos**

La siguiente función `sc` es un ejemplo que demuestra el uso del modificador `static` en el lenguaje de programación C++.

```
int* sc(int* p, int l, int d) {
  static int *a;
  static int al;

  if (p != NULL and l != -1) {
    al = l;
    a = p;
  }

  while (al > 0 and not *a != d) {
    al--;
    a++;
  }

  if (al == 0) return NULL;

  al--;
  return a++;
}
```

La función `sc` es utilizada de una manera muy extraña. La primera vez que es invocada, los argumentos los dos primeros argumentos tienen la dirección válida de un arreglo de enteros y su longitud, el tercer argumento muestra un valor que es utilizado; las siguientes invocaciones los dos primeros argumentos deben ser `NULL` y una longitud negativa.

Explique que hace la función `sc`.

Nombre: _____

Código: _____

Indicación: Sea breve en sus respuestas y justifiquelas concisamente. Excepto las preguntas de selección múltiple.

1. (25 points) **Aplicación de BNF en DSL**[1]

   El BNF ha sido utilizado no solamente para definir lenguajes de programación sino también para definir ficheros[2] de configuración de aplicaciones como por ejemplo el muy conocido servidor *Web* apache.

   La figura 1 muestra un BNF que describe un DSL para que los profesores generen parciales autómaticamente, la descripción del BNF es auto contenida.

$$
\begin{array}{rcl}
\langle \text{Parciales} \rangle & ::= & \langle \text{Parcial} \rangle \\
& | & \langle \text{Parcial} \rangle \langle \text{Parciales} \rangle \\
\langle \text{Parcial} \rangle & ::= & \textit{IdParcial}\ '['\ \langle \text{ListaPuntos} \rangle\ ']' \\
\langle \text{ListaPuntos} \rangle & ::= & \langle \text{Punto} \rangle \\
& | & \langle \text{Punto} \rangle\ ';'\ \langle \text{ListaPuntos} \rangle \\
\langle \text{Punto} \rangle & ::= & \textit{IdPunto}\ '\{'\ \langle \text{Atributos} \rangle\ '\}' \\
\langle \text{Atributos} \rangle & ::= & \epsilon \\
& | & \langle \text{Atributo} \rangle \\
& | & \langle \text{Atributo} \rangle\ ':'\ \langle \text{Atributos} \rangle \\
\langle \text{Atributo} \rangle & ::= & \text{pregunta facil} \\
& | & \text{pregunta dificil} \\
& | & \text{pregunta imposible} \\
& | & \text{seleccion multiple} \\
& | & \text{seleccion unica} \\
& | & \text{seleccion oculta}
\end{array}
$$

   Figura 1: DSL para generación automática de parciales

   Donde el *IdParcial* es “Parcial 1”, “Parcial 2”, etc., donde el *IdPunto* es “Punto 1, Punto 2”, etc.

   a) Escriba un fichero de configuración para un curso con 2 parciales, donde cada parcial tiene 2 puntos con preguntas fáciles y de selección múltiple.
   b) Escriba un fichero de configuración para un curso con 3 parciales, con dos puntos con preguntas imposibles.

[1] Domain Specific Language
[2] Archivos

Solution:

1. ```
   Parcial1 [
      Punto1 {
         pregunta facil : seleccion multiple
      }
      ;
      Punto2 {
         pregunta facil : seleccion multiple
      }]
   Parcial2 [
      Punto1 {
         pregunta facil : seleccion multiple
      }
      ;
      Punto2 {
         pregunta facil : seleccion multiple
      }]
   ```

2. ```
   Parcial1 [
      Punto1 {
         pregunta imposible
      }]
   Parcial2 [
      Punto1 {
         pregunta imposible
      }]
   Parcial3 [
      Punto1 {
         pregunta imposible
      }]
   ```

2. (25 points) **EWE**
Escriba un programa en EWE que lea tres números a, b y c. El programa debe iterar acumulando los valores de a y b hasta que dicho acumulador sea superior a 10 veces el valor de c. El programa debe imprimir el número de iteraciones.

$$\sum_{i=1}^{i=n} (a+b) > (10 \times c)$$

Solution:

```
# Punto-02-a
readInt(a)
readInt(b)
readInt(c)
uno := 1
diez := 10
sum := a + b
mult := diez * c
n := uno
acum := sum
loop:
if acum > mult then goto end
n := n + uno
acum := acum + sum
goto loop
end: writeInt(n)
halt
equ a M[0]
equ b M[1]
equ c M[2]
equ acum M[3]
equ mult M[4]
equ n M[5]
equ uno M[6]
equ sum M[7]
equ diez M[8]
```

3. (25 points) **Gramáticas independientes de contexto**
En la figura 2

$$
\begin{aligned}
V &= \{A, B, C\} \\
\Sigma &= \{\dashv, \vdash\} \\
P &= \{A \rightarrow B \mid C \mid \epsilon, B \rightarrow \dashv\dashv\ A\ \dashv, C \rightarrow \vdash\ A\ \vdash\vdash\} \\
S &= A
\end{aligned}
$$

Figura 2: Gramática independiente de contexto

a) Construya la derivación para obtener la frase: ⊣⊣⊢⊢⊢⊣.

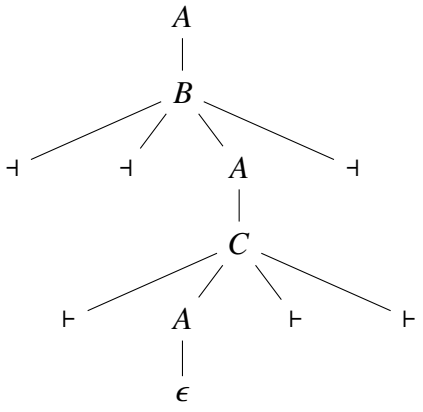
b) Construya el árbol de derivación de la derivación del punto anterior.

Solution:

a)

$$
\begin{aligned}
A &\Rightarrow B \\
&\Rightarrow \dashv\dashv\ A\ \dashv \\
&\Rightarrow \dashv\dashv\ C\ \dashv \\
&\Rightarrow \dashv\dashv\vdash\ A\ \vdash\vdash\dashv \\
&\Rightarrow \dashv\dashv\vdash\ \epsilon\ \vdash\vdash\dashv \\
&\equiv \dashv\dashv\vdash\vdash\vdash\dashv
\end{aligned}
$$

b)



4. (25 points) **Otras gramáticas: Diagrama sintáctico**

   Usted ha sido contratado para probar un nuevo video juego diseñado por la famosa empresa de videos juegos Boring Inc. Es conocido que cada problema de computación puede ser descrito por una gramática, la casa matriz ha diseñado el video juego utilizando la gramática que se describe en la figura 3.

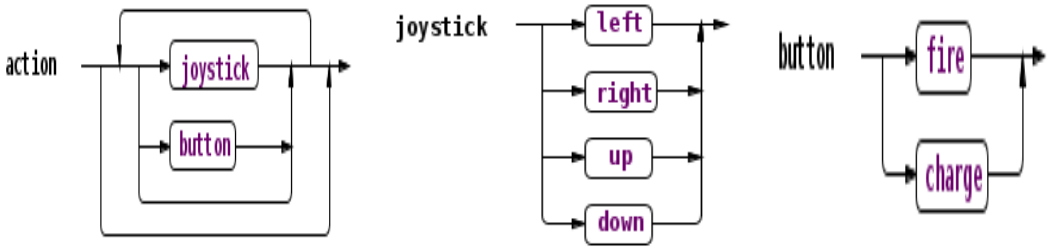


Figura 3: Diagrama sintáctico para un video juego

Escriba programa que utilizando la anterior gramática que permita matar los dos monstruos que hay en un campo de batalla. Suponga que el primer monstruo está ubicado en $x = 2, y = 2$ y el segundo está ubicado en $x = 3, y = 1$. Usted se encuentra ubicado en la posición $x = 0, y = 0$. Tenga en cuenta: que el arma debe ser cargada antes de disparar y que el arma dispara por contacto es decir en la misma posición donde está ubicado el monstruo.

Solution:
Una posible solución:

```
right
right
up
up
charge
fire
right
down
charge
fire
```

ST0244 Lenguajes de programación 031-032-033 **Universidad EAFIT**
Primer Parcial (Tema B) **DIS**
Juan Francisco Cardona McCormick **7 de marzo de 2014**

Nombre: _____

Código: _____

Indicación: Sea breve en sus respuestas y justifiquelas concisamente. Excepto las preguntas de selección múltiple.

1. (25 points) **Aplicación de BNF en DSL**[1]

   El BNF ha sido utilizado no solamente para definir lenguajes de programación sino también para definir ficheros[2] de configuración de aplicaciones como por ejemplo el muy conocido servidor *Web* apache.

   La figura 1 muestra un BNF que describe un DSL para que los profesores generen parciales autómaticamente, la descripción del BNF es auto contenida.

$$
\begin{array}{rcl}
\langle \text{Parciales} \rangle & ::= & \langle \text{Parcial} \rangle \\
& | & \langle \text{Parcial} \rangle \langle \text{Parciales} \rangle \\
\langle \text{Parcial} \rangle & ::= & \textit{IdParcial}\ '['\ \langle \text{ListaPuntos} \rangle\ ']' \\
\langle \text{ListaPuntos} \rangle & ::= & \langle \text{Punto} \rangle \\
& | & \langle \text{Punto} \rangle\ ';'\ \langle \text{ListaPuntos} \rangle \\
\langle \text{Punto} \rangle & ::= & \textit{IdPunto}\ '\{'\ \langle \text{Atributos} \rangle\ '\}' \\
\langle \text{Atributos} \rangle & ::= & \epsilon \\
& | & \langle \text{Atributo} \rangle \\
& | & \langle \text{Atributo} \rangle\ ':'\ \langle \text{Atributos} \rangle \\
\langle \text{Atributo} \rangle & ::= & \text{pregunta facil} \\
& | & \text{pregunta dificil} \\
& | & \text{pregunta imposible} \\
& | & \text{seleccion multiple} \\
& | & \text{seleccion unica} \\
& | & \text{seleccion oculta}
\end{array}
$$

   Figura 1: DSL para generación automática de parciales

   Donde el *IdParcial* es “Parcial 1”, “Parcial 2”, etc., donde el *IdPunto* es “Punto 1, Punto 2”, etc.

   a) Escriba un fichero de configuración para un curso con 2 parciales, donde cada parcial tiene 3 puntos con preguntas fáciles y de selección múltiple.
   b) Escriba un fichero de configuración para un curso con 2 parciales, con dos puntos con preguntas imposibles.

[1] Domain Specific Language
[2] Archivos

Solution:

1. ```
   Parcial1 [
       Punto1 {
          pregunta facil : seleccion multiple
       }
       ;
       Punto2 {
          pregunta facil : seleccion multiple
       }
       ;
       Punto3 {
          pregunta facil : seleccion multiple
       }]
     Parcial2 [
       Punto1 {
          pregunta facil : seleccion multiple
       }
       ;
       Punto2 {
          pregunta facil : seleccion multiple
       }
       ;
       Punto3 {
          pregunta facil : seleccion multiple
       }]
   ```

2. ```
   Parcial1 [
       Punto1 {
          pregunta imposible
       }
       ;
       Punto2 {
          pregunta imposible
       }]
     Parcial2 [
       Punto1 {
          pregunta imposible
       }
       ;
       Punto2 {
          pregunta imposible
       }]
   ```

2. (25 points) **EWE**
Escriba un programa en EWE que lea tres números a, b, c, d. El programa debe iterar acumulando la diferencia de a y b hasta que dicho acumulador sea superior a c veces el valor de d. El programa debe imprimir el número de iteraciones. (**Nota:** los valores de las variables son positivos)

$$\sum_{i=1}^{i=n} (a - b) > (c \times d)$$

Solution:

```
# Punto-03-b
readInt(a)
readInt(b)
readInt(c)
readInt(d)
diff := a - b
mult := c * d
uno := 1
n := uno
sum := diff
loop:
if sum > mult then goto end
n := n + uno
sum := sum + diff
goto loop
end: writeInt(n)
halt
equ a M[0]
equ b M[1]
equ c M[2]
equ d M[3]
equ diff M[4]
equ n M[5]
equ uno M[6]
equ sum M[7]
equ mult M[8]
```

3. (25 points) **Gramáticas independientes de contexto**
En la figura 2

$$
\begin{aligned}
V &= \{A, B, C\} \\
\Sigma &= \{\dashv, \vdash\} \\
P &= \{A \rightarrow B \mid C \mid \epsilon, B \rightarrow \dashv\dashv\ A\ \dashv, C \rightarrow \vdash\ A\ \vdash\vdash\} \\
S &= A
\end{aligned}
$$

Figura 2: Gramática independiente de contexto

a) Construya la derivación para obtener la frase: ⊣⊣⊢⊢⊢⊣.

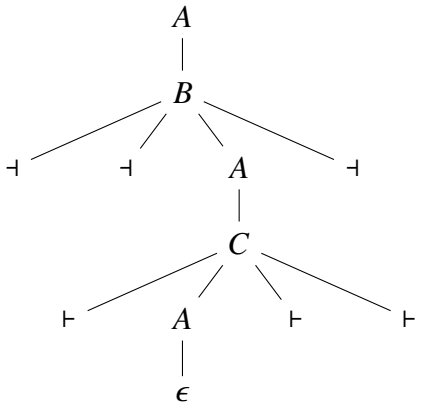
b) Construya el árbol de derivación de la derivación del punto anterior.

Solution:

a)

$$
\begin{aligned}
A &\Rightarrow B \\
&\Rightarrow \dashv\dashv\ A\ \dashv \\
&\Rightarrow \dashv\dashv\ C\ \dashv \\
&\Rightarrow \dashv\dashv\vdash\ A\ \vdash\vdash\dashv \\
&\Rightarrow \dashv\dashv\vdash\ \epsilon\ \vdash\vdash\dashv \\
&\equiv \dashv\dashv\vdash\vdash\vdash\dashv
\end{aligned}
$$

b)



4. (25 points) **Otras gramáticas: Diagrama sintáctico**

   Usted ha sido contratado para probar un nuevo video juego diseñado por la famosa empresa de videos juegos Boring Inc. Es conocido que cada problema de computación puede ser descrito por una gramática, la casa matriz ha diseñado el video juego utilizando la gramática que se describe en la figura 3.

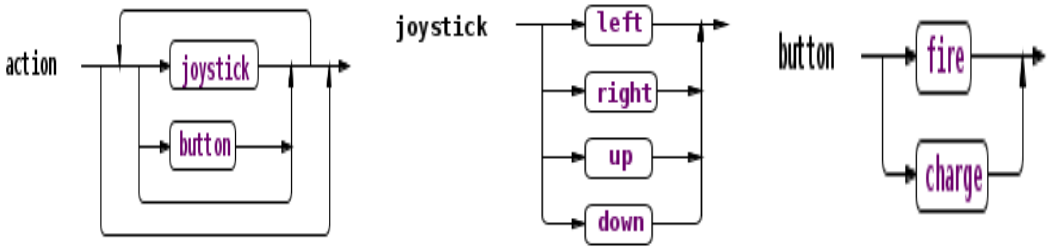


Figura 3: Diagrama sintáctico para un video juego

Escriba programa que utilizando la anterior gramática que permita matar los dos monstruos que hay en un campo de batalla. Suponga que el primer monstruo está ubicado en $x = 1, y = 2$ y el segundo está ubicado en $x = 4, y = 1$. Usted se encuentra ubicado en la posición $x = 0, y = 0$. Tenga en cuenta: que el arma debe ser cargada antes de disparar y que el arma dispara por contacto es decir en la misma posición donde está ubicado el monstruo.

Solution:
Una posible solución:

```
right
right
up
up
charge
fire
down
right
right
right
charge
fire
```

Nombre: _____
Código: _____

1. (15 points) **Tipos de datos**
   El siguiente programa no compila por que tiene obviamente algunos errores. ¿Podría usted indicar cuántos errores tiene el siguiente programa? ¿Y cuál es la razón de cada error?

```
1  #include <iostream>
2
3  using namespace std;
4
5  int
6  main() {
7    double a;
8    float b;
9    int c;
10   char d;
11
12   a = 72.10f;
13   b = 72.20d;
14   c = 'a';
15   d = 0x48;
16
17   cout << " a: " << a << " b: " << b << " c: " << c << " d: " << d
18        << endl;
19   return 0;
20 }
```

Solution: La anomalía se observa en la línea 13 del programa:

```
b = 72.20d;
```

El problema es que se está asignando un double a un flotante, por lo tanto se esta guardando un tipo muy grande (double) en un tipo más pequeño.

El problema es que los compiladores no son estándares:

Compilando el programa en cygwin se obtiene:

```
$ make tiposDatos
g++     tiposDatos.cpp   -o tiposDatos
$
```

En Cygwin no hay problema.

Pero compilando el programa en Mac se obtiene:

```
c++     tiposDatos.cpp   -o tiposDatos
tiposDatos.cpp:18:12: error: invalid suffix 'd' on floating constant
  b = 72.20d;
           ^
```

2. (15 points) **Tipos de datos en C++ y argumentos de funciones**

   Dado el siguiente código cuáles son los valores de *x*, *y* y *z* al final de la ejecución del programa. Muestre la ejecución paso a paso.

```
#include <iostream>
using namespace std;

int f(int a, int b, int c) { return ++a + b++ + ++c; }

int g(int a, int b, int c) { return a++ + ++b + c++; }

int h(int a, int b, int c) { return ++a + b++ + ++c; }

int main() {
  int x = f(1,1,1);
  int y = g(x,x,x);
  int z = h(x,y,1);

  cout << " x: " << x << " y: " << y << " z: " << z << endl;
  return 0;
}
```

Solution: Invocación de `f(1,1,1);`

```
a <- 1
b <- 1
c <- 1
```

Evalúa la instrucción y retorna:

```
return ++a + b++ + ++c;
return (2) + (1) + (2);
```

En la función `main`:

```
x <- 5
```

En `main` se llama la función `g(x,x,x);` En `g` los parámetros reales quedan:

```
a <- 5
b <- 5
c <- 5
```

Evalúa la instrucción:

```
return a++ + ++b + c++;
return 5 + 6 + 5;
```

El valor de retorno queda en 16. Por lo tanto en `main`:

```
y <- 16
```

En `main` se evalúa la siguiente instrucción: `h(x,y,1);`, la evalución pasa los parámetros formales `h(5,16,1)`. En `h` los parámetros formales quedan:

```
a <- 5
b <- 16
c <- 1
```

Al evaluar en `h` la instrucción:

```
return ++a + b++ + ++c;
return 6 + 16 + 2;
```

Retorna 24, `z <- 24` la salida queda:

```
" x: " 5  " y: " 16 " z: " 24
```

3. (15 points) **Ámbito**
   Muestre paso a paso cual es valor de salida de siguiente programa:

```
#include <iostream>

using namespace std;

int i;

int p(int y) {
  int j = y;
  i++;
  return j+y;
}

void q(void) {
  int j = 2;
  i = 0;
  i = p(i = i + j);
}

int
main() {
  q();
  cout << i << endl;
  return 0;
}
```

Solution:

```
global: i <- 0 // Inicialización
main()
   q();
      int j = 2; // local: j <- 2
      i = 0; // global: i <- 0
      p (i = i + j); // p ( global: i <- 0 + 2 => i <- 2) => p(2)
      p(); // local: y <- 2
         int j = y; // local: j <- y => j <- 2
         i++; // global: i <- i++ => i <- 3
         return j+y; // valor a retornar <- j + y => 2 + 2 => 4
      i = p(); // return global i <- 4
   cout << i << endl; // Imprime 4
-- Termino
```

4. (15 points) **Ámbito y apuntadores**
   Mostrar paso a paso como obtiene el valor que imprime el siguiente programa.

```
#include <iostream>
using namespace std;

int* f(int *x, int *y) {
  int *tmp = x;
  x = y;
  y = tmp;
  if (*x > *y) return y;  else return x;
}

int* g(int *x, int *y) {
  int *tmp = x;
  x = y;
  y = tmp;
  if (*x < *y) return  y; else return x;
}

int main() {
  int a = 10, b= 20, c= 30, d = 40;
  int *p = g(f(g(&a,&b),g(&c,&d)), f(g(&b,&c),g(&a,&d)));
  cout << *p << endl;
}
```

Solution:

```
main:
  // local a <- 10 b <- 20 c <- 30 d <- 40
  g(&a,&b);
     // x -> main:a, y -> main:b
     int *tmp = x; // tmp es un apuntador que apunta a x => tmp -> main:a
     x = y; // x <- y => x -> main:b
     y = tmp; // y <- tmp => y -> main:a
     if (*x < *y) return  y; else return x;
     // (*x < *y) => main:b < main:a => 20 < 10 => false
     // retorna x => main:b
  g(&c,&d);
     // x -> main:c, y -> main:d
     int *tmp = x; // tmp es un apuntador que apunta a x => tmp -> main:c
     x = y; // x <- y => x -> main:d
     y = tmp; // y <- tmp => y -> main:c
     if (*x < *y) return  y; else return x;
     // (*x < *y) => main:d < main:c => 40 < 30 => false
```

```
    // retorna x => &main:d
f(&main:b,&main:d);
    // x -> main:b, y -> main:d
    int *tmp = x; // tmp es un apuntador que apunta a x => tmp -> main:b
    x = y; // x <- y => x -> main:d
    y = tmp; // y <- tmp => y -> main:b
    if (*x > *y) return y;  else return x;
    // (*x > *y) => main:d > main:b => 40 > 20 => true
    // returna y => &main:b
 g(&b,&c);
    // x -> main:b, y -> main:c
    int *tmp = x; // tmp es un apuntador que apunta a x => tmp -> main:b
    x = y; // x <- y => x -> main:c
    y = tmp; // y <- tmp => y -> main:b
    if (*x < *y) return  y; else return x;
    // (*x < *y) => main:c < main:b => 30 < 20 => false
    // retorna x => main:c
 g(&a,&d);
    // x -> main:a, y -> main:d
    int *tmp = x; // tmp es un apuntador que apunta a x => tmp -> main:a
    x = y; // x <- y => x -> main:d
    y = tmp; // y <- tmp => y -> main:a
    if (*x < *y) return  y; else return x;
    // (*x < *y) => main:d < main:a => 40 < 10 => false
    // retorna x => &main:d
f(&main:c,&main:d);
    // x -> main:c, y -> main:d
    int *tmp = x; // tmp es un apuntador que apunta a x => tmp -> main:c
    x = y; // x <- y => x -> main:d
    y = tmp; // y <- tmp => y -> main:c
    if (*x > *y) return y;  else return x;
    // (*x > *y) => main:d > main:c => 40 > 30 => true
    // returna y => &main:c
 g(&main:c,&main:c);
    // x -> main:c, y -> main:d
    int *tmp = x; // tmp es un apuntador que apunta a x => tmp -> main:c
    x = y; // x <- y => x -> main:c
    y = tmp; // y <- tmp => y -> main:c
    if (*x < *y) return  y; else return x;
    // (*x < *y) => main:d < main:c => 30 < 30 => false
    // retorna x => &main:c
 int *p = ...// p -> main:c
```

```
    cout << *p << endl; // Imprime 30.
```

5. (30 points) **Funciones, ciclos e implementación**
   La siguiente es la signatura de las dos siguiente funciones que se encuentra en el fichero `interfaz.h`:

```
#pragma once

int f1(int a, int b, int c, int d);

bool f2(int a, int b, int c);

int f3(int a, int b);

int f4(int a);
```

Donde **`f4`** es una función que calcula la suma de los valores desde 1 hasta *a inclusive*. La función **`f3`** calcula la suma de los valores desde *a* a *b inclusive*. La función **`f2`** computa las sumatoria de los valores desde *a* a *b inclusive*, de *a* + 1 a *b inclusive*, ..., *b* a *b inclusive* y verifica si el valor *c* es computado por cada una de esas computaciones intermedias. **`f1`** computa la sumatoria de los valores desde *a* a *b inclusive*, de *a* + 1 a *b inclusive*, ..., *b* a *b inclusive* y verifica si el valor *c* o *d* es computado por cada una de ellas, si *c* es computado retorna *c*, si *d* es computado retorna *d*, si ambos son computados retorna la suma de ambos valores, en caso contrario retorna cero.

Escriba la implementación de cada una de las funciones anteriores en un fichero llamado `implementación.cpp`.

Solution:

```
int f1(int a, int b, int c, int d) {
  if (f2(a,b,c))
    return c
  else if (f2(a,b,d))
    return d
  else return 0;
}

bool f2(int a, int b, int c) {
  int i;
  bool encontro = false;

  for (i = a; !encontro && a <= b; i++) {
    if (f3(i,b) == c)
      encontro = true;
  }

  return encontro;
}

```

```
21 int f3(int a, int b) {
22
23   return f4(b) - f4(a) + 1;
24 }
25
26 int f4(int a) {
27   int sum = 0;
28   for (int i = 1; i <= a; i++) {
29     sum += i;
30   }
31   return sum;
32 }
```

6. (10 points) **Uso de funciones y la utilidad *make***
Suponga que usted ya implementó las funciones del punto 5 y las quiere utilizar en un programa llamado **principal.cpp**. Este programa principal lee cuatro valores *a*, *b*, *c* y *d* y muestra el resultado de la invocación de la función `f1`.

   a) Escriba el programa completo de `principal.cpp`.

   b) Escriba el contenido de un fichero `Makefile` que permite crear un ejecutable llamado *principal*.

Solution:

a) Programa `principal.cpp`:

```cpp
#include "interfaz.h"
#include <iostream>

using namespace std;

int
main() {

  int a, b, c, d;

  cin >> a >> b >> c >> d;

  cout << f1(a,b,c,d) << endl;

  return 0;
}
```

b) `Makefile`:

```
principal: principal.o interfaz.o
	g++ -o principal principal.o

principal.o: principal.cpp interfaz.h
	g++ -c principal.cpp

interfaz.o: interfaz.cpp interfaz.h
	g++ -c interfaz.cpp
```

Nombre: _____
Código: _____

1. (15 points) **Tipos de datos**
   El siguiente programa no compila por que tiene obviamente algunos errores. ¿Podría usted indicar cuántos errores tiene el siguiente programa? ¿Y cuál es la razón de cada error?

```
1  #include <iostream>
2  using namespace std;
3
4  int
5  main() {
6    float a;
7    double b;
8    int c;
9    char d;
10
11   a = 72.10d;
12   b = 72.20f;
13   c = 'a';
14   d = 0x48;
15
16   cout << " a: " << a << " b: " << b << " c: " << c << " d: " << d
17        << endl;
18
19   return 0;
20 }
```

Solution: En total: Un sólo error.

El problema está en la línea 10, no se puede poner un valor double en un valor float.

En Mac el resultado de la compilación es igual a:

```
make -k tiposDatos2
c++     tiposDatos2.cpp   -o tiposDatos2
tiposDatos2.cpp:17:12: error: invalid suffix 'd' on floating constant
  a = 72.10d;
           ^
1 error generated.
```

En cygwin la compilación:

```
$ make tiposDatos2
g++     tiposDatos2.cpp   -o tiposDatos2
$
```

El programa compila.

2. (15 points) **Tipos de datos en C++ y argumentos de funciones**

   Dado el siguiente código cuáles son los valores de *x*, *y* y *z* al final de la ejecución del programa. Muestre la ejecución paso a paso.

```
#include <iostream>
using namespace std;

int f(int a, int b, int c) { return ++a + ++b + ++c; }

int g(int a, int b, int c) { return a++ + b++ + c++; }

int h(int a, int b, int c) { return ++a + b++ + ++c; }

int
main() {
  int x = f(1,2,1);
  int y = g(x,x,x);
  int z = h(x,y,1);

  cout << " x: " << x << " y: " << y << " z: " << z << endl;
}
```

Solution: Invocación de `f(1,2,1);`

```
a <- 1
b <- 2
c <- 1
```

Evalúa la instrucción y retorna:

```
return ++a + ++b + ++c;
return (2) + (3) + (2);
```

En la función `main`:

```
x <- 7
```

En `main` se llama la función `g(x,x,x);` En `g` los parámetros reales quedaN:

```
a <- 7
b <- 7
c <- 7
```

Evalúa la instrucción:

```
return a++ + b++ + c++;
return 7 + 7 + 7;
```

El valor de retorno queda en 21. Por lo tanto en `main`:

```
y <- 21
```

En `main` se evalúa la siguiente instrucción: `h(x,y,1);`, la evalución pasa los parámetros formales `h(7,21,1)`. En `h` los parámetros formales quedan:

```
a <- 7
b <- 21
c <- 1
```

Al evaluar en `h` la instrucción:

```
return ++a + b++ + ++c;
return 8 + 21 + 2;
```

Retorna 31, `z <- 31` la salida queda:

```
" x: " 7  " y: " 21 " z: " 31
```

3. (15 points) **Ámbito**
   Muestre paso a paso cual es valor de salida de siguiente programa:

```
#include <iostream>

using namespace std;

int i;

int p(int y) {
  int j = y;
  i++;
  return j+y;
}

void q(void) {
  int j = 4;
  i = 0;
  i = p(i += j);
}

main() {
  q();
  cout << i << endl;
  return 0;
}
```

Solution:

```
global: i <- 0 // Inicialización
main()
   q();
     int j = 4; // local: j <- 4
     i = 0; // global: i <- 0
     p (i += j); // p (global: i <-0 + 4 => i <- 4) => p(4)
     p(); // local y <- 4
        int j = y; // loca: j <- y => j <- 4
        i++; // global i <- i++ => i <- 5
        return j+y; // valor a retornar <- j + y => 4 + 4 => 8
     i = p(); // return global i <- 8
   cout << i << endl; // Imprime 8
-- Termino
```

4. (15 points) **Ámbito y apuntadores**
   Mostrar paso a paso como obtiene el valor que imprime el siguiente programa.

```
#include <iostream>
using namespace std;

int* f(int *x, int *y) {
  int *tmp = x;
  x = y;
  y = tmp;
  if (*x > *y) return y; else return x;
}

int* g(int *x, int *y) {
  int *tmp = x;
  x = y;
  y = tmp;
  if (*x < *y) return  y; else return x;
}

int main() {
  int a = 10, b= 20, c= 30, d = 40;
  int *p = f(g(f(&a,&b),f(&c,&d)), f(g(&b,&c),f(&a,&d)));
  cout << *p << endl;
}
```

Solution:

```
main:
  // local a <- 10 b <- 20 c <- 30 d <- 40
  f(&a,&b);
     // x -> main:a, y -> main:b
     int *tmp = x; // tmp es un apuntador que apunta a x => tmp -> main:a
     x = y; // x <- y => x -> main:b
     y = tmp; // y <- tmp => y -> main:a
     if (*x > *y) return y;  else return x;
     // (*x > *y) => main:b > main:a => 20 > 10 => true
     // retorna y => &main:a
  f(&c,&d);
     // x -> main:c, y -> main:d
     int *tmp = x; // tmp es un apuntador que apunta a x => tmp -> main:c
     x = y; // x <- y => x -> main:d
     y = tmp; // y <- tmp => y -> main:c
     if (*x > *y) return y;  else return x;
     // (*x > *y) => main:d > main:c => 40 > 30 => true
```

```
    // retorna y => &main:c
  g(&main:a,&main:c)
    // x -> main:a, y -> main:c
    int *tmp = x; // tmp es un apuntador que apunta a x => tmp -> main:a
    x = y; // x <- y => x -> main:c
    y = tmp; // y <- tmp => y -> main:a
    if (*x < *y) return  y; else return x;
    // (*x < *y) => main:c < main:a => 30 < 10 => false
    // retorna x => &main:c
  g(&b,&c)
    // x -> main:b, y -> main:c
    int *tmp = x; // tmp es un apuntador que apunta a x => tmp -> main:b
    x = y; // x <- y => x -> main:c
    y = tmp; // y <- tmp => y -> main:b
    if (*x < *y) return  y; else return x;
    // (*x < *y) => main:c < main:b => 30 < 20 => false
    // retorna x => &main:c
  f(&a,&d);
    // x -> main:a, y -> main:d
    int *tmp = x; // tmp es un apuntador que apunta a x => tmp -> main:a
    x = y; // x <- y => x -> main:d
    y = tmp; // y <- tmp => y -> main:a
    if (*x > *y) return y;  else return x;
    // (*x > *y) => main:d > main:a => 40 > 10 => true
    // retorna y => &main:a
  f(&main:c,&main:a);
    // x -> main:c, y -> main:a
    int *tmp = x; // tmp es un apuntador que apunta a x => tmp -> main:c
    x = y; // x <- y => x -> main:c
    y = tmp; // y <- tmp => y -> main:a
    if (*x > *y) return y;  else return x;
    // (*x > *y) => main:a > main:d => 10 > 40 => false
    // retorna y => &main:a
  f(&main:c,&main:a);
    // x -> main:c, y -> main:a
    int *tmp = x; // tmp es un apuntador que apunta a x => tmp -> main:c
    x = y; // x <- y => x -> main:c
    y = tmp; // y <- tmp => y -> main:a
    if (*x > *y) return y;  else return x;
    // (*x > *y) => main:a > main:d => 10 > 40 => false
    // retorna y => &main:a
  int *p = ...// p -> main:a
```

```
    cout << *p << endl; // Imprime 10.
```

5. (30 points) **Funciones, ciclos e implementación**
   La siguiente es la signatura de las dos siguiente funciones que se encuentra en el fichero `interfaz.h`:

```
#pragma once

int f1(int a, int b, int c, int d);

bool f2(int a, int b, int c);

int f3(int a, int b);

int f4(int a);
```

Donde **`f4`** es una función que calcula la suma de los valores desde 1 hasta *a inclusive*. La función **`f3`** calcula la suma de los valores desde *a* a *b inclusive*. La función **`f2`** computa las sumatoria de los valores desde *a* a *b inclusive*, de *a* + 1 a *b inclusive*, ..., *b* a *b inclusive* y verifica si el valor *c* es computado por cada una de esas computaciones intermedias. **`f1`** computa la sumatoria de los valores desde *a* a *b inclusive*, de *a* + 1 a *b inclusive*, ..., *b* a *b inclusive* y verifica si el valor *c* o *d* es computado por cada una de ellas, si *c* es computado retorna *c*, si *d* es computado retorna *d*, si ambos son computados retorna la suma de ambos valores, en caso contrario retorna cero.

Escriba la implementación de cada una de las funciones anteriores en un fichero llamado `implementación.cpp`.

Solution:

```
int f1(int a, int b, int c, int d) {
  if (f2(a,b,c))
    return c
  else if (f2(a,b,d))
    return d
  else return 0;
}

bool f2(int a, int b, int c) {
  int i;
  bool encontro = false;

  for (i = a; !encontro && a <= b; i++) {
    if (f3(i,b) == c)
      encontro = true;
  }

  return encontro;
}

```

```
21 int f3(int a, int b) {
22
23   return f4(b) - f4(a) + 1;
24 }
25
26 int f4(int a) {
27   int sum = 0;
28   for (int i = 1; i <= a; i++) {
29     sum += i;
30   }
31   return sum;
32 }
```

6. (10 points) **Uso de funciones y la utilidad *make***
Suponga que usted ya implementó las funciones del punto 5 y las quiere utilizar en un programa llamado **principal.cpp**. Este programa principal lee cuatro valores *a*, *b*, *c* y *d* y muestra el resultado de la invocación de la función `f1`.

   a) Escriba el programa completo de `principal.cpp`.

   b) Escriba el contenido de un fichero `Makefile` que permite crear un ejecutable llamado *principal*.

Solution:

a) Programa `principal.cpp`:

```
#include "interfaz.h"
#include <iostream>

using namespace std;

int
main() {

  int a, b, c, d;

  cin >> a >> b >> c >> d;

  cout << f1(a,b,c,d) << endl;

  return 0;
}
```

b) `Makefile`:

```
principal: principal.o interfaz.o
	g++ -o principal principal.o

principal.o: principal.cpp interfaz.h
	g++ -c principal.cpp

interfaz.o: interfaz.cpp interfaz.h
	g++ -c interfaz.cpp
```

Nombre: _____
Código: _____

1. (25 puntos) **Ascendente**
   La función `ascendente` tiene la siguiente signatura:

```
enum Dir { nodir, asc, desc };

DirFun ascendente(const char *cadena);
```

   Donde `Dir` es una enumeración que indica la dirección de crecimiento de una cadena de caracteres terminada con nulo: `asc` la cadena crece ascendentemente, por ejemplo:

```
cadena[] = "abcde"; // Ascedente
cadena[] = "acefij"; // Acendente
cadena[] = ""; // Ascendete
cadena[] = "a"; // Ascendete
```

   Pero esta función no es estricta, la siguiente cadenas son ascendentes:

```
cadena[] = "aaaaa";  // Ascendete
cadena[] = "aabbbccccccc"; // Ascendete
```

   O no tiene una dirección determinada:

```
cadena[] = "acbfajla"; // No Ascendete
```

   O crece en dirección descendente:

```
cadena[] = "zyxwv";  // No Ascendente
cadena[] = "zxvpm";  // No Ascendente
```

   La función `ascendente` debe indicar si una cadena crece ascendentemente o no lo hace: retorna `asc` o `nodir`.

   Implemente la correspondiente función en C++.

Solution: La siguiente es una posible solución para la solución de ascendentes:

```
1  enum Dir { nodir, asc, desc };
2
3  Dir ascendente(const char *cadena) {
4
5    if (!cadena[0]) return asc;
6    if (!cadena[1]) return asc;
7
8    int diff = 0;
9
10   if ((diff = (cadena[1] - cadena[0])) < 0) return nodir;
11
12   for (int i = 2; cadena[i] and
13           ((diff = (cadena[i] - cadena[i-1])) >= 0); i++);
14
15   if (diff < 0) return nodir;
16   return asc;
17 }
```

2. (25 puntos) **Interfaces, implementación y C++**
   En la figura 1 se observa una definición de un interfaz llamada `IPerilla`. Esta interfaz define las operaciones básicas que se hacen con este tipo de dispositivos: girar a la izquierda (disminuir en una cantidad) o girar a la derecha (aumentar en una cantidad) y obtener su valor actual. Con esta interfaz se puede implementar muchos tipos de perillas por ejemplo para las perillas de lavadoras (ciclo suave, ciclo fuerte, estregar, etc.). En este caso vamos a implementar una perilla discreta de 10 posiciones (`PerillaDiscreta10`. La perilla es un contador circular: comienza en 0 y cuando alcanza su nivel máximo 10 al ser incrementado vuelve a 0, y al contrario cuando esta en 0 y es disminuido pasa a 10. El argumento pasado a las funciones `giroIzquierda` y `giroDerecha` ***siempre*** es un valor entre 0 y 10.

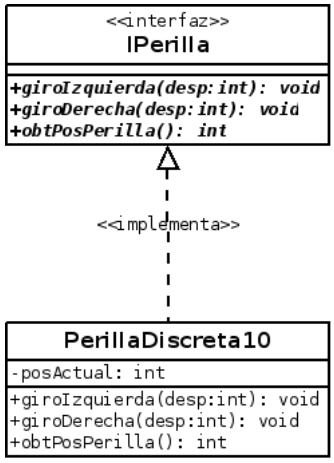


Figura 1: Interfaz e implementación

En este punto se van a escribir las clases del diagrama de clases en C++. Haciendo la declaración de la interfaz `IPerilla` y la clase `PerillaDiscreta10` así como la definición de métodos.

Solution: Una posible solución.

Declaración de las clases `IPerilla`, `PerillaDiscreta10`:

```
class IPerilla {
 public:
  virtual void giroDerecha(int desp) = 0;
  virtual void giroIzquierda(int desp) = 0;
  virtual int  obtPosPerilla() const = 0;
};

class PerillaDiscreta10 : public IPerilla {
 private:
  int posActual;
 public:
  PerillaDiscreta10();
  void giroIzquierda(int desp);
  void giroDerecha(int desp);
  int obtPosPerilla() const;
 private:
  int compDesp(float desp);
};
```

Implementación de los métodos de la clase `PerillaDiscreta10`:

```
PerillaDiscreta10::PerillaDiscreta10() : posActual(0) { }

void PerillaDiscreta10::giroDerecha(int desp) {

  posActual = (posActual + desp) % 11;
}

void PerillaDiscreta10::giroIzquierda(int desp) {

  posActual = (posActual - desp) % 11;
}

int PerillaDiscreta10::obtPosPerilla() const {

  return posActual;
}

ManejadorPerillas::ManejadorPerillas() {
  izq = new PerillaDiscreta10();
  der = new PerillaDiscreta10();
}

ManejadorPerillas::~ManejadorPerillas() {
  try { delete izq; delete der; } catch (...) { }
}
```

```
26
27 PerillaDiscreta10* ManejadorPerillas::obtPerillaIzq() const {
28   return izq;
29 }
30
31 PerillaDiscreta10* ManejadorPerillas::obtPerillaDer() const {
32   return der;
33 }
```

3. (25 puntos) `Clase abstractas en C++`

   En la figura 2 se muestra un diagrama de clases en la cual se observan dos nuevas clases : `ManejadorPerillas` y `ManejadorPerillasP`. La primera clase es una clase abstracta que tiene dos perillas de tipo `PerillaDiscreta10` una a la izquierda y otra la derecha, ésta clase representa es también una instancia de tipo `IPerilla`. La segunda clase `ManejadorPerillasP` representa el comportamiento de ambas perillas trabajando en paralelo, es decir si la operación mover a la izquierda es aplicada a la clase, ambas perillas (`izq` y `der`) se mueven a la izquierda; igualmente si la operación que se aplica es `girarDerecha` ambas perillas se le aplica a cada una la operación de `girarDerecha` con el mismo desplazamiento. La operación `obtPosPerilla` obtiene la suma de ambas perillas.

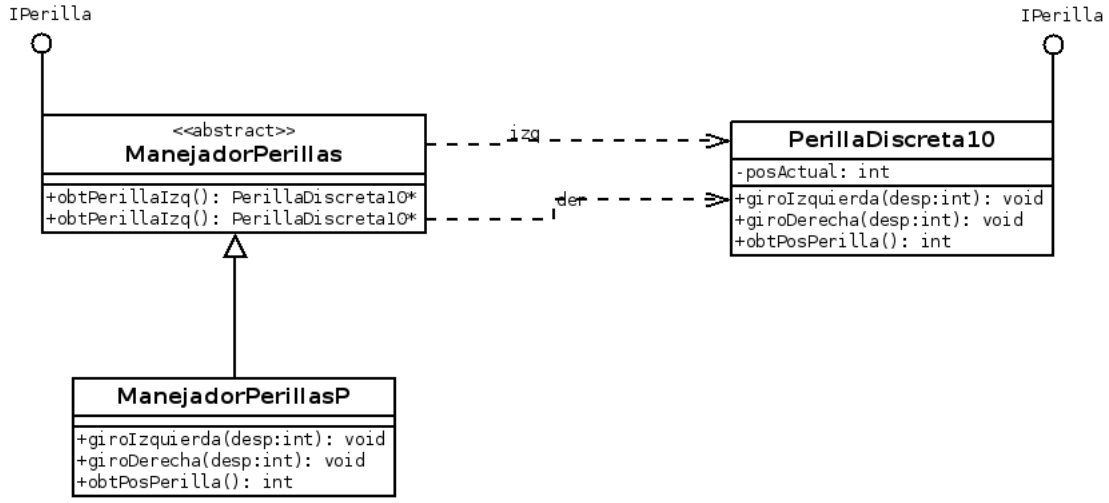


Figura 2: Abstracción y herencia

Escriba la declaración de ambas clases: `ManejadorPerillas` y `ManejadorPerillasP`; escriba la implementación de los métodos de ambas clases.

Solution: Una posible solución. Declaración de la clases: `ManejadorPerillas` y `ManejadorPerillasP`.

```
class ManejadorPerillas : public IPerilla {
 private:
  PerillaDiscreta10 *izq;
  PerillaDiscreta10 *der;
 public:
  ManejadorPerillas();
  ~ManejadorPerillas();
  PerillaDiscreta10* obtPerillaIzq() const;
  PerillaDiscreta10* obtPerillaDer() const;
};

class ManejadorPerillasP : public ManejadorPerillas {
 public:
  ManejadorPerillasP();
  void giroIzquierda(int desp);
  void giroDerecha(int desp);
  int obtPosPerilla() const;
};
```

Implementación de los métodos de las clases: `ManejadorPerilla` y `ManejadorPerillaP`:

```
ManejadorPerillasP::ManejadorPerillasP()  : ManejadorPerillas() { }

void ManejadorPerillasP::giroIzquierda(int desp) {
  this->obtPerillaIzq()->giroIzquierda(desp);
  this->obtPerillaDer()->giroIzquierda(desp);
}

void ManejadorPerillasP::giroDerecha(int desp) {
  this->obtPerillaIzq()->giroDerecha(desp);
  this->obtPerillaDer()->giroDerecha(desp);
}

int ManejadorPerillasP::obtPosPerilla() const {
  return this->obtPerillaIzq()->obtPosPerilla() +
    this->obtPerillaDer()->obtPosPerilla();
}
```

4. (25 puntos) **Herencia en C++**

   Dado el siguiente código en C++:

```
1  class A {
2  public:
3       A() : { }
4       virtual void p() { std::cout << "A.p" << std::endl; }
5       void q() { std::cout << "A.q" << std::endl; }
6       virtual void r() { p(); q(); }
7  };
8
9  class B : public A {
10 public:
11      B() : A() { }
12      virtual void p() { std::cout << "B.p" << std::endl; }
13 };
14
15 class C : public B {
16 public:
17    C() : B() { }
18    void q() { std::cout << "C.q" << std::endl; }
19    void r() { q(); p(); }
20 };
21
22 int main() {
23   ...
24   A a;
25   B b;
26   a = b;
27   a.r();
28   A* ap = new C;
29   ap->r();
30   ap = new B;
31   ap->r();
32   ...
33 }
```

   ¿Qué es lo que imprime el código? ¿Por qué?

Solution: Vamos paso a paso. En la línea 24, se crea una instancia de la clase `A`. En la línea 25, se crea un instancia de la clase `B`. En la línea 26, se hace una asignación de la instancia de `B` a una instancia de `A`, esto solamente copia los atributos que son comunes en la clase `A`, la variable `a` sigue siendo una instancia de la clase `A`. En la línea 27, se llama al método `r()` de la clase `A`, por lo tanto invoca primero al método `p()` de la clase `A` y luego al método `q()` de la clase `A`. La salida queda:

```
A.p
A.q
```

En la línea 28, se declara una variable `ap` de tipo apuntador a `A`, pero se inicializa con una dirección de tipo `C`. En la línea 29, se llama al método `r()` en un instancia de clase `C`, , esta invoca los métodos `q()`, en la clase `C` y el método `p()`, que no se encuentra definido en la clase **B**:

```
C.q
B.p
```

En la línea 30, se asigna a la variable `ap` una dirección de un tipo **B**, el objeto que apunta es de tipo B aunque este asignado a una variable de tipo dirección de `A`. En la línea 31, se llama al método `r()`, pero como esta clase es de tipo **B** esta llama al método sobrecargado en la clase **A**, que invoca primero al método `p()` y luego a `q()`, pero como en la clase `C` solamente esta sobrecargado `q()`, se llama primero a este método y luego a `p()` en la clase padre que primer lo tenga sobrecargado esto es en la clase `B`, así que la salida queda así:

```
B.p
A.q
```

Resumiendo la salida queda así:

```
A.p
A.q
C.q
B.p
B.p
A.q
```

Nombre: _____
Código: _____

1. (25 puntos) **Descendente**
   La función `descendente` tiene la siguiente signatura:

```
enum Dir { nodir, asc, desc };

DirFun desscendente(const char *cadena);
```

   Donde `Dir` es una enumeración que indica la dirección de crecimiento de una cadena de caracteres terminada con nulo: `asc` la cadena crece descendentemente, por ejemplo:

```
cadena[] = "zyxwv"; // Descedente
cadena[] = "zxwncba"; // Descendente
cadena[] = ""; // Descendete
cadena[] = "z"; // Descendete
```

   Pero esta función no es estricta, la siguiente cadenas son descendentes:

```
cadena[] = "zzzzz";  // Descendente
cadena[] = "zzzzyyyywwwwaaaa"; // Descendente
```

   O no tiene una dirección determinada:

```
cadena[] = "zaybw"; // No Descendente
```

   O crece en dirección ascendente:

```
cadena[] = "abcde";  // No Descendente
cadena[] = "aaabbcc";  // No Descendente
```

   La función `descendente` debe indicar si una cadena decrece o no lo hace: retorna `desc` o `nodir`.

   Implemente la correspondiente función en C++.

Solution: Una posible solución.

```
Dir descendente(const char *cadena) {

  if (!cadena[0]) return desc;
  if (!cadena[1]) return desc;

  int diff = 0;

  if ((diff = (cadena[1] - cadena[0])) > 0) return nodir;

  for (int i = 2; cadena[i] and
        ((diff = (cadena[i] - cadena[i-1])) <= 0); i++);

  if (diff > 0) return nodir;
  return desc;
}
```

2. (25 puntos) **Interfaces, implementación y C++**
En la figura 1 se observa una definición de un interfaz llamada `IPerilla`. Esta interfaz define las operaciones básicas que se hacen con este tipo de dispositivos: girar a la izquierda (disminuir en una cantidad) o girar a la derecha (aumentar en una cantidad) y obtener su valor actual. Con esta interfaz se puede implementar muchos tipos de perillas por ejemplo para las perillas de lavadoras (ciclo suave, ciclo fuerte, estregar, etc.). En este caso vamos a implementar una perilla discreta de 5 posiciones (`PerillaDiscreta5`. La perilla es un contador circular: comienza en 0 y cuando alcanza su nivel máximo 5 al ser incrementado vuelve a 0, y al contrario cuando esta en 0 y es disminuido pasa a 5. El argumento pasado a las funciones `giroIzquierda` y `giroDerecha` ***siempre*** es un valor entre 0 y 5.

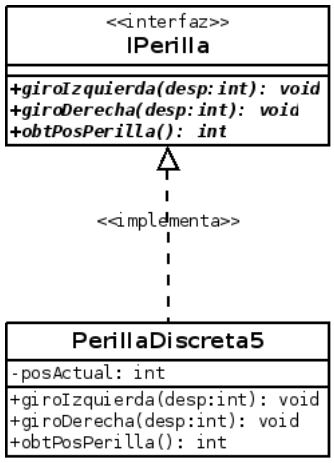


Figura 1: Interfaz e implementación

En este punto se van a escribir las clases del diagrama de clases en C++. Haciendo la declaración de la interfaz `IPerilla` y la clase `PerillaDiscreta5` así como la definición de métodos.

Solution: Una posible solución:

```
1
2  class IPerilla {
3   public:
4    virtual void giroDerecha(int desp) = 0;
5    virtual void giroIzquierda(int desp) = 0;
6    virtual int  obtPosPerilla() const = 0;
7  };
8
9  class PerillaDiscreta5 : public IPerilla {
10  private:
11   int posActual;
12  public:
13   PerillaDiscreta5();
14   void giroIzquierda(int desp);
```

```
15    void giroDerecha(int desp);
16    int obtPosPerilla() const;
17   private:
18    int compDesp(float desp);
19  };
```

3. (25 puntos) `Clase abstractas en C++`

   En la figura 2 se muestra un diagrama de clases en la cual se observan dos nuevas clases : `ManejadorPerillas` y `ManejadorPerillasD`. La primera clase es una clase abstracta que tiene dos perillas de tipo `PerillaDiscreta5` una a la izquierda y otra la derecha, esta clase representa es también una instancia de tipo `IPerilla`. La segunda clase `ManejadorPerillasD` representa el comportamiento de ambas perillas trabajando en secuencia: es decir que si la operación mover a la derecha es aplicada a la clase: se incrementa primera la perilla que está al lado derecho, en el moto solicitado[1], si ésta perilla genera un ***overflow***, es decir pasa de 5 a 0 (o más), se mueve la perilla del lado izquierdo hacia la derecha en una unidad; si la operación es de movimiento a la izquierda se mueve la perilla de la derecha el monto solicitado[2] y se verifica si existe un ***underflow***, es decir pasa de 0 a 5, se mueve la perilla del lado izquierdo hacia la izquierda una unidad. La operación `obtPosPerilla` obtiene aplicando la siguiente formula:

   $$izq.obtPosPerilla * 5 + der.obtPosPerilla$$

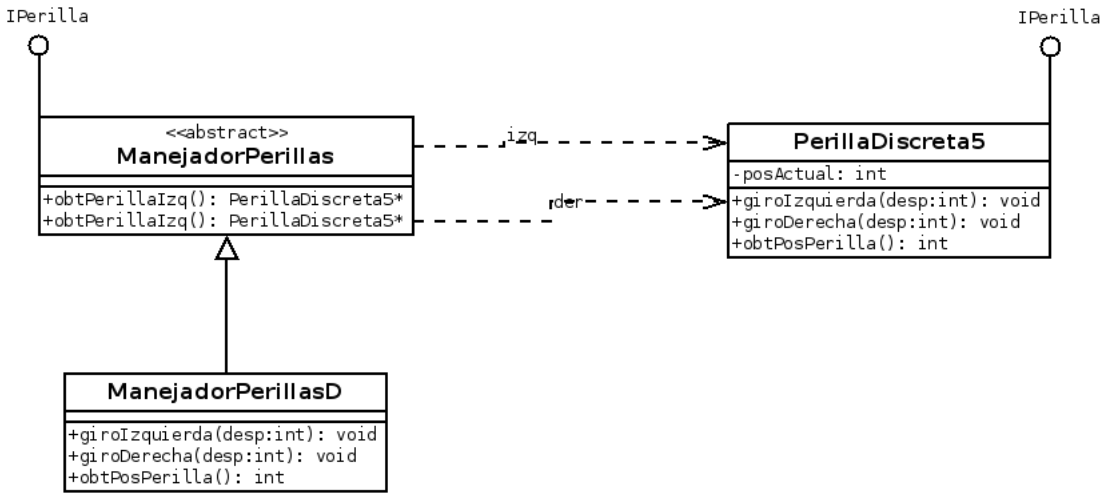


Figura 2: Abstracción y herencia

Escriba la declaración de ambas clases: `ManejadorPerillas` y `ManejadorPerillasP`; escriba la implementación de los métodos de ambas clases.

[1]Este valor esta siempre entre 0 y 5
[2]Este valor está entre 0 y 5

Solution: Declaración de las clases: `ManejadorPerillas` y `ManejadorPerillasP`:

```
class ManejadorPerillas : public IPerilla {
 private:
  PerillaDiscreta5 *izq;
  PerillaDiscreta5 *der;
 public:
  ManejadorPerillas();
  ~ManejadorPerillas();
  PerillaDiscreta5* obtPerillaIzq() const;
  PerillaDiscreta5* obtPerillaDer() const;
};

class ManejadorPerillasD : public ManejadorPerillas {
 public:
  ManejadorPerillasD();
  void giroIzquierda(int desp);
  void giroDerecha(int desp);
  int obtPosPerilla() const;
};
```

Ahora la solución de las implementaciones de los métodos de ambas clases:

```
ManejadorPerillas::ManejadorPerillas() {
  izq = new PerillaDiscreta5();
  der = new PerillaDiscreta5();
}

ManejadorPerillas::~ManejadorPerillas() {
  try { delete izq; delete der; } catch (...) { }
}

PerillaDiscreta5* ManejadorPerillas::obtPerillaIzq() const {
  return izq;
}

PerillaDiscreta5* ManejadorPerillas::obtPerillaDer() const {
  return der;
}

ManejadorPerillasD::ManejadorPerillasD()  : ManejadorPerillas() { }

void ManejadorPerillasD::giroIzquierda(int desp) {

  if (this->obtPerillaDer()->obtPosPerilla() - desp < 0) {
    this->obtPerillaIzq()->giroIzquierda(1);
  }
  this->obtPerillaDer()->giroIzquierda(desp);
}

```

```
28 void ManejadorPerillasD::giroDerecha(int desp) {
29   if (this->obtPerillaDer()->obtPosPerilla() - desp < 0) {
30     this->obtPerillaIzq()->giroDerecha(1);
31   }
32
33   this->obtPerillaDer()->giroDerecha(desp);
34 }
35
36 int ManejadorPerillasD::obtPosPerilla() const {
37   return this->obtPerillaIzq()->obtPosPerilla() * 5 +
38     this->obtPerillaDer()->obtPosPerilla();
39 }
```

4. (25 puntos) **Herencia en C++**

   Dado el siguiente código en C++:

```
1  class A {
2  public:
3       A() : { }
4       virtual void p() { std::cout << "A.p" << std::endl; }
5       void q() { std::cout << "A.q" << std::endl; }
6       virtual void r() { p(); q(); }
7  };
8
9  class B : public A {
10 public:
11      B() : A() { }
12      virtual void p() { std::cout << "B.p" << std::endl; }
13 };
14
15 class C : public B {
16 public:
17    C() : B() { }
18    void q() { std::cout << "C.q" << std::endl; }
19    void r() { q(); p(); }
20 };
21
22 int main() {
23   ...
24    A a;
25    B b;
26    a = b;
27    a.r();
28    A* ap = new C;
29    ap->r();
30    ap = new B;
31    ap->r();
32   ...
33 }
```

   ¿Qué es lo que imprime el código? ¿Por qué?

Solution: Vamos paso a paso. En la línea 18, se crea una instancia de la clase `A`. En la línea 19, se crea un instancia de la clase `C`. En la línea 20, se hace una asignación de la instancia de `C` a una instancia de `A`, esto solamente copia los atributos que son comunes en la clase `A`, la variable `a` sigue siendo una instancia de la clase `A`. En la línea 21, se llama al método `r()` de la clase `A`, por lo tanto invoca primero al método `p()` de la clase `A` y luego al método `q()` de la clase `A`. La salida queda:

```
A.p
A.q
```

En la línea 22, se declara una variable `ap` de tipo apuntador a `A`, pero se inicializa con una dirección de tipo `B`. En la línea 23, se llama al método `r()` en un instancia de clase `B`, pero esta no tiene definido este método así que busca en la clase padre `A`, esta invoca los métodos `p()`, en la clase `B` y el método `q()`, que no se encuentra definido en la clase **B**, así que llama al método en la clase padre `A`, así que la salida queda así:

```
B.p
A.q
```

En la línea 24, se asigna a la variable `ap` una dirección de un tipo **C**, el objeto que apunta es de tipo B aunque este asignado a una variable de tipo dirección de `A`. En la línea 25, se llama al método `r()`, pero como esta clase es de tipo **C** esta llama al método sobrecargado en la clase **C**, que invoca primero al método `q()` y luego a `p()`, pero como en la clase `C` solamente esta sobrecargado `q()`, se llama primero a este método y luego a `p()` en la clase padre que primer lo tenga sobrecargado esto es en la clase `B`, así que la salida queda así:

```
C.q
B.q
```

Resumiendo la salida queda así:

```
A.p
A.q
B.p
A.q
C.q
B.q
```