**Laboratorio 3**

Estructuras de Datos Avanzadas

Universidad Nacional de San Agustín

Nombre: Alexander Rusvell Apaza Torres

1. Transformar expresión postfija a un árbol de
   1. Código fuente:
      1. Nodo padre: De este nodo heredarán 2 hijos: iNodo y sNodo. De esta manera podremos obtener el resultado de la expresión con la función *getValue().*

class Nodo

{

protected:

Nodo\* izq;

Nodo\* der;

int id;

public:

Nodo();

virtual int getValue() = 0;

virtual void getRawValue() = 0;

virtual string getStrValue() = 0;

~Nodo();

};

Nodo::Nodo()

{

this->der = 0;

this->izq = 0;

}

* + 1. iNodo: Este nodo almacenará los números de la expresión. Hereda de Nodo.

class iNodo : public Nodo

{

private:

int value;

public:

iNodo(int value);

int getValue() override;

void getRawValue() override;

string getStrValue() override;

~iNodo();

friend class Arbol;

};

iNodo::iNodo(int value)

{

this->value = value;

}

int iNodo::getValue(){

return this->value;

}

void iNodo::getRawValue(){

cout<<this->value;

}

string iNodo::getStrValue(){

return to\_string(this->value);

}

* + 1. sNodo: Este nodo almacenará los operadores de la expresión. También hereda de Nodo.

class sNodo : public Nodo {

private:

string value;

public:

sNodo(string value);

int getValue() override;

void getRawValue() override;

string getStrValue() override;

~sNodo();

friend class Arbol;

};

sNodo::sNodo(string value){

this->value = value;

}

int sNodo::getValue(){

if (this->value == "+")

return this->izq->getValue() + this->der->getValue();

else if (this->value == "-")

return this->izq->getValue() - this->der->getValue();

else if (this->value == "\*")

return this->izq->getValue() \* this->der->getValue();

}

void sNodo::getRawValue(){

cout<<this->value;

}

string sNodo::getStrValue(){

return this->value;

}

* + 1. Árbol: Aquí se genera el arbol de expresión. Recibe una lista con los elementos de la expresión en notación postfija y la transforma en un árbol de expresión. También generá un .dot para observar el árbol en sí y su resultado.

class Arbol {

private:

Nodo\* root;

list<Nodo\*> pila;

public:

Arbol(list<string>);

void showStack();

void showR(Nodo\*, int);

void show();

void generateDot(string&, Nodo\*);

void setIDs(Nodo\* t, int&, string&);

string getDot();

~Arbol();

};

Arbol::Arbol(list<string> post){

int v;

string vs;

while (!post.empty()){

if(isdigit(post.front()[0])){

v = stoi(post.front());

pila.push\_back(new iNodo(v));

post.pop\_front();

}

else{

vs = post.front();

Nodo\* nuevo = new sNodo(vs);

nuevo->der = this->pila.back();

this->pila.pop\_back();

nuevo->izq = this->pila.back();

this->pila.pop\_back();

pila.push\_back(nuevo);

post.pop\_front();

}

}

}

void Arbol::generateDot(string& res, Nodo\* t) {

if (!t) return;

if (t->izq) res = res + t->id + " -> " + t->izq->id + "; \n";

if (t->der) res = res + t->id + " -> " + t->der->id + "; \n";

generateDot(res, t->izq);

generateDot(res, t->der);

}

void Arbol::setIDs(Nodo\* t, int& i,string& res) {

if (!t) return;

t->id = i;

res = res + i + " [ label = \"" + t->getStrValue() + "\" ]; \n";

i = i + 1;

setIDs(t->izq,i, res);

setIDs(t->der,i, res);

}

string Arbol::getDot() {

string res = "digraph G {\n";

int i = 0;

this->setIDs(this->root, i, res);

this->generateDot(res, this->root);

res = res + "Resultado -> " + this->root->getValue() + "; \n";

res = res + "}";

return res;

}

* + 1. Resultado: El programa en sí imprime el .dot, pero se captura la salida en un archivo de texto y luego se ejecuta el comando para generar la imagen

int main(int argc, char const \*argv[]) {

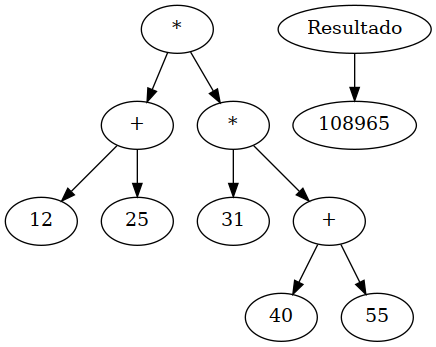
list<string> post = {"12","25","+","31","40","55","+","\*","\*"};

Arbol c(post);

cout<<c.getDot()<<endl;

return 0;

}



* 1. Mostrar un árbol binario usando graphViz:
     1. Código:

template<typename T>

class Arbol

{

private:

Nodo<T>\* root;

public:

Arbol(){

this->root = 0;

}

void push(T value, Nodo<T>\*& r){

if (!r){

r = new Nodo<T>(value);

return;

}

else{

push(value,r->children[value > r->value ]);

}

}

void push(T value){

this->push(value, this->root);

}

void generateDot(string& res, Nodo<T>\* t){

if (!t) return;

if (t->children[0]){

res = res + t->value + " -> " + t->children[0]->value + "; \n";

if (t->children[1]) res = res + to\_string(t->value) + " -> " + to\_string(t->children[1]->value) + "; \n";

generateDot(res, t->children[0]);

generateDot(res, t->children[1]);

}

string getDot(){

string res = "digraph G {\n";

this->generateDot(res, this->root);

res = res + "}";

return res;

}

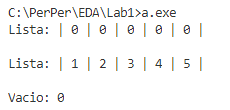
};

* + 1. Demostración:

int main()

{

VLista<5> a;

a.show();

a.push(1);

a.push(2);

a.push(3);

a.push(4);

a.push(5);

a.push(4);

cout<<endl;

a.show();

cout<<endl<<"Vacio: "<< a.isEmpty()<<endl;

return 0;

}

* 1. VPila
     1. Código

template <int N>

class VPila : public VStructure<N>

{

public:

VPila(){}

~VPila(){}

int pop(){

int res;

this->actual--;

res = this->array[this->actual];

this->array[this->actual]=0;

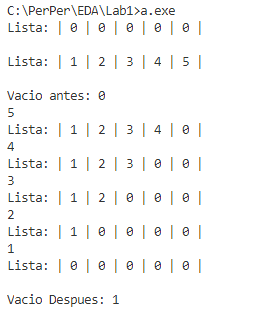
return res;

}

};

* + 1. Demostración

int main()

{

VPila<5> a;

a.show();

a.push(1);

a.push(2);

a.push(3);

a.push(4);

a.push(5);

a.push(4);

cout<<endl;

a.show();

cout<<endl<<"Vacio antes: "<< a.isEmpty()<<endl;

cout<<a.pop()<<endl;

a.show();

cout<<a.pop()<<endl;

a.show();

cout<<a.pop()<<endl;

a.show();

cout<<a.pop()<<endl;

a.show();

cout<<a.pop()<<endl;

a.show();

cout<<endl<<"Vacio Despues: "<< a.isEmpty()<<endl;

return 0;

}

* 1. VCola
     1. Código

template<int N>

class VCola : public VStructure<N>

{

public:

VCola(){}

int pop()

{

if (this->actual==0)

return -1;

int res= this->array[0];

for(int i = 1; i < this->actual ; i++){

this->array[i-1]=this->array[i];

}

this->array[this->actual-1]=0;

this->actual--;

return res;

}

~VCola(){

}

};

* + 1. Demostración

int main()

{

VCola<5> a;

a.show();

a.push(1);

a.push(2);

a.push(3);

a.push(4);

a.push(5);

a.push(4);

cout<<endl;

a.show();

cout<<endl<<"Vacio antes: "<< a.isEmpty()<<endl;

cout<<a.pop()<<endl;

a.show();

cout<<a.pop()<<endl;

a.show();

cout<<a.pop()<<endl;

a.show();

cout<<a.pop()<<endl;

a.show();

cout<<a.pop()<<endl;

a.show();

cout<<endl<<"Vacio Despues: "<< a.isEmpty()<<endl;

return 0;

}

1. Con Punteros
   1. PStructure: Esta clase será padre de las clases PLista, PCola y PPila
      1. Código
         1. Clase PNodo

class PNodo

{

private:

int value;

PNodo\* siguiente;

public:

PNodo(int );

~PNodo();

friend class PStructure;

};

PNodo::PNodo(int v)

{

this->value=v;

this->siguiente=0;

}

PNodo::~PNodo()

{

}

* + - 1. Clase PStructure

class PStructure

{

protected:

PNodo\* first;

PNodo\* last;

public:

PStructure(){

this->first=0;

this->last=0;

}

void push(int v){

PNodo\* nuevo=new PNodo(v);

if (last){

last->siguiente=nuevo;

last=nuevo;

return;

}

first=last=nuevo;

return;

}

void show()

{

PNodo\* temp= this->first;

cout<<"Lista: | ";

while(temp){

cout<<temp->value<<" | ";

temp=temp->siguiente;

}

cout<<endl;

}

void remove(int position){

PNodo\* temp=this->first;

if (position==0){

this->first=this->first->siguiente;

}

for(int i = 0; i < position-1; i++){

temp=temp->siguiente;

}

cout<<endl<<"Dato Borrado: "<<temp->value<<endl;

if (!temp){

return;

}

if (temp->siguiente && temp->siguiente != this->last){

temp->siguiente=temp->siguiente->siguiente;

}

else if(temp->siguiente && temp->siguiente==last){

temp->siguiente=0;

}

else if(!temp->siguiente){

this->first=0;

this->last=0;

}

}

~PStructure(){}

};

* 1. Clase PLista
     1. Código:

class PLista : public PStructure

{

public:

PLista(){}

~PLista(){}

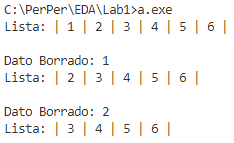
};

* + 1. Demostración:

int main()

{

PLista a;

a.push(1);

a.push(2);

a.push(3);

a.push(4);

a.push(5);

a.push(6);

a.show();

a.remove(0);

a.show();

a.remove(0);

a.show();

}

* 1. Clase PCola
     1. Código

class PCola : public PStructure

{

public:

PCola(){}

int pop()

{

PNodo\* temp=this->first;

PStructure::remove(0);

return temp->value;

}

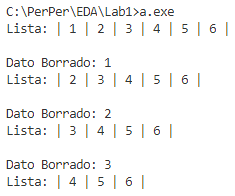
~PCola(){}

};

* + 1. Demostración

int main()

{

PCola a;

a.push(1);

a.push(2);

a.push(3);

a.push(4);

a.push(5);

a.push(6);

a.show();

a.pop();

a.show();

a.pop();

a.show();

a.pop();

a.show();

}

* 1. Clase PPila
     1. Código

class PPila : public PStructure

{

public:

PPila(){}

int pop()

{

PNodo\* temp= this->first;

while(temp->siguiente!= this->last){

temp=temp->siguiente;

}

int res=temp->siguiente->value;

temp->siguiente=0;

this->last=temp;

cout<<endl<<"Dato Borrado: "<<res<<endl;

return res;

}

~PPila(){}

};

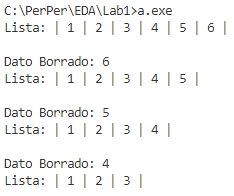
* + 1. Demostración

int main()

{

PPila a;

a.push(1);

a.push(2);

a.push(3);

a.push(4);

a.push(5);

a.push(6);

a.show();

a.pop();

a.show();

a.pop();

a.show();

a.pop();

a.show();

}