Listas

List.h

```
#ifndef LISTSTACKQUEUE_LIST_H
#define LISTSTACKQUEUE_LIST_H

#include <iostream>
#include <cassert>

using namespace std;

template<typename T>
struct Node{
    T data;
    Node<T>* next;
};

template<typename T>
class List {
private:
    Node<T>* begin;
    int count;
    Node<T>* makeNode(const T& value);
public:
    List();
    ~List();
    void insert(int pos, const T& value);
    void erase(int pos);
    T& get(int pos) const;
    void print() const;
    int size() const;
};
#endif //LISTSTACKQUEUE_LIST_H
```

List.cpp

```
#ifndef LISTSTACKQUEUE_LIST_CPP
#define LISTSTACKQUEUE_LIST_CPP

#include "List.h"

template<typename T>
List<T>:: List(): begin(0), count(0){

}
template<typename T>
List<T>:: ~List(){
    Node<T>* del = begin;
    while (begin){
        begin = begin->next;
        delete del;
```

```
        del = begin;
    }
}
template<typename T>
Node<T>* List<T>::makeNode(const T &value) {
    Node<T>* temp = new Node<T>;
    temp->data = value;
    temp->next = 0;
    return temp;
}
template<typename T>
void List<T>::insert(int pos, const T &value) {
    if(pos < 0 || pos>count){
        cout << "Error! The position is out of range." << endl;
        return;
    }
    Node<T>* add = makeNode(value);
    if(pos == 0){
        add->next = begin;
        begin = add;
    }else{
        Node<T>* cur = begin;
        for(int i=0; i<pos-1; i++){
            cur = cur->next;
        }
        add->next = cur->next;
        cur->next = add;
    }
    count++;
}

template<typename T>
void List<T>::erase(int pos) {
    if(pos < 0 || pos>count){
        cout << "Error! The position is out of range." << endl;
        return;
    }
    if(pos == 0){
        Node<T>* del = begin;
        begin = begin->next;
        delete del;
    }else{
        Node<T>* cur = begin;
        for(int i=0; i<pos-1; i++){
            cur = cur->next;
        }
        Node<T>* del = cur->next;
        cur->next = del->next;
        delete del;
    }
    count--;
}
template<typename T>
T& List<T>::get(int pos) const{
    if(pos < 0 || pos>count-1){
```

```
        cout << "Error! The position is out of range." << endl;
        assert(false);
    }
    if(pos == 0){
        return begin->data;
    }else{
        Node<T>* cur = begin;
        for(int i=0; i<pos; i++){
            cur = cur->next;
        }
        return cur->data;
    }
}
template<typename T>
void List<T>::print() const{
    if(count == 0){
        cout << "List is empty." << endl;
        return;
    }
    Node<T>* cur = begin;
    while(cur){
        cout << cur->data << " ";
        cur = cur->next;
    }
}
template<typename T>
int List<T>::size() const {
    return count;
}
#endif
```

main.cpp

```
#include "List.cpp"

using namespace std;
int main() {
    List<string> list;
    list.insert(0, "nestor");
    list.insert(1, "victor");
    list.insert(2, "Maria");
    list.insert(3, "juan");
    list.insert(4, "pedro");
    list.print();
    cout << "\nSize: " << list.size() << endl;
    cout << "Element (2): " << list.get(2) << endl;

    list.erase(2);
    list.erase(3);
    list.print();
    cout << "\nSize: " << list.size() << endl;
    return 0;
```

```
}
```

Pilas

Stack.h

```
#ifndef LISTSTACKQUEUE_STACK_H
#define LISTSTACKQUEUE_STACK_H

#include "List.cpp"

template<typename T>
class Stack {
private:
    List<T> list;
public:
    void push(const T& value);
    void pop();
    T& top() const;
    int size() const;
};
#endif //LISTSTACKQUEUE_STACK_H
```

Stack.cpp

```
#ifndef LISTSTACKQUEUE_STACK_CPP
#define LISTSTACKQUEUE_STACK_CPP

#include "Stack.h"

template<typename T>
void Stack<T>::push(const T &value) {
    list.insert(0, value);
}

template<typename T>
void Stack<T>::pop(){
    list.erase(0);
}

template<typename T>
T& Stack<T>::top() const {
    return list.get(0);
}

template<typename T>
int Stack<T>::size() const {
    return list.size();
}
#endif
```

main.cpp

```
#include "Stack.cpp"

using namespace std;
int main() {
   Stack<string> stack;
   stack.push("Colombia");
   stack.push("Ecuador");
   stack.push("Venezuela");

   cout << "Pila" << endl;
   cout << "Tamanio pila: " << stack.size() << endl;
   cout << "Ultimo elemento: " << stack.top() << endl;
   while(stack.size() > 0){
       cout << "Elemento top: " << stack.top() << endl;
       stack.pop();
   }
   cout << "Tamanio pila: " << stack.size() << endl;
   return 0;
}
```

Convertidor de HEX a decimal

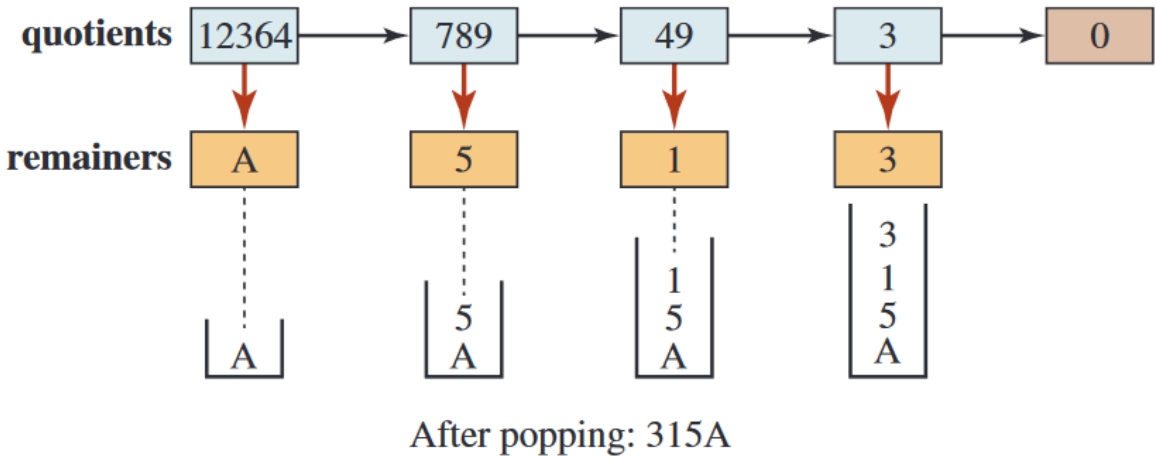

Figure 18.13 Converting a decimal number to a hexadecimal number

main.cpp

```
#include "Stack.cpp"

using namespace std;
int main() {
   Stack<char> pila;
```

```
    string convertidor = "0123456789ABCDEF";
    string resultado = "";
    int decimal = 0;
    int index = 0;

    cout << "Ingrese numero decimal: ";
    cin >> decimal;

    // Convertir decimal a hexadecimal y agregar a la pila
    while(decimal != 0){
        int residuo = decimal % 16;
        pila.push(convertidor[residuo]);
        decimal = decimal / 16;
    }

    // llenar resultado inversamente hexadecimal usando la pila
    while(pila.size() > 0){
        resultado.push_back(pila.top());
        pila.pop();
    }

    cout << "Resultado: " << resultado;
    return 0;
}
```

Invertir texto

main.cpp

```
#include "Stack.cpp"

using namespace std;
int main() {
    Stack<char> pila;
    string origin;
    string reversed;

    cout << "Ingrese frase a invertir: ";
    getline(cin, origin);
    for(char c: origin){
        pila.push(c);
    }

    //Estraer elementos de la pila y agregarlos al string
    while(pila.size() > 0){
        reversed.push_back(pila.top());
        pila.pop();
    }

    cout << "Original: " << origin << endl;
    cout << "Invertido: " << reversed << endl;

    return 0;
```

```
}
```

Balanceo de paréntesis

main.cpp

```cpp
#include "Stack.cpp"

using namespace std;

bool verificarParentesis(string expresion){
    Stack<char> pila;
    bool balanceado = true;
    int indice = 0;
    while(indice < expresion.size() && balanceado){
        char simbolo = expresion[indice];
        if(simbolo == '('){
            pila.push(simbolo);
        }else{
            if(pila.size() == 0){
                balanceado = false;
            }else{
                pila.pop();
            }
        }
        indice++;
    }
    if(balanceado && pila.size() == 0){
        return true;
    }else{
        return false;
    }
}

int main() {

    string expresion;

    cout << "Ingresar expresion: ";
    cin >> expresion;

    cout << boolalpha << verificarParentesis(expresion);

    return 0;
}
```

Balanceo de símbolos

main.cpp

```
#include "Stack.cpp"

using namespace std;

bool parejas(char simboloApertura, char simboloCierre){
   string aperturas = "([{";
   string cierres = ")]}";
   return aperturas.find(simboloApertura) ==
cierres.find(simboloCierre);
}

bool verificarSimbolos(string expresion){
   Stack<char> pila;
   bool balanceado = true;
   int indice = 0;
   while(indice < expresion.size() && balanceado){
       char simbolo = expresion[indice];
       if(simbolo == '(' || simbolo == '[' || simbolo == '{'){
           pila.push(simbolo);
       }else{
           if(pila.size() == 0){
               balanceado = false;
           }else{
               char tope = pila.top();
               pila.pop();
               if(!parejas(tope, simbolo)){
                   balanceado = false;
               }
           }
       }
       indice++;
   }
   if(balanceado && pila.size() == 0){
       return true;
   }else{
       return false;
   }
}

int main() {

   cout << boolalpha << verificarSimbolos("{({[][]})()}") <<
endl;
   cout << boolalpha << verificarSimbolos("[{()]") << endl;
   return 0;
}
```

Colas

Queue.h

```
#ifndef COLAS_QUEUE_H
#define COLAS_QUEUE_H

#include "List.h"

template<typename T>
class Queue {
private:
   List<T> list;
public:
   void push(const T& value);
   void pop();
   T& front() const;
   T& back() const;
   int size() const;
   void print() const;
};


#endif //COLAS_QUEUE_H
```

Queue.cpp

```
#ifndef COLAS_QUEUE_CPP
#define COLAS_QUEUE_CPP

#include "Queue.h"


template<typename T>
void Queue<T>::push(const T &value) {
   list.insert(list.size(), value);
}

template<typename T>
void Queue<T>::pop(){
   list.erase(0);
}

template<typename T>
T& Queue<T>::front() const {
   return list.get(0);
}

template<typename T>
T& Queue<T>::back() const {
   return list.get(list.size()-1);
}

template<typename T>
int Queue<T>::size() const {
   return list.size();
}
```

```
template<typename T>
void Queue<T>::print() const{
   list.print();
}

#endif
```

main.cpp

```
#include <iostream>
#include "List.cpp"
#include "Queue.cpp"

using namespace std;
int main() {
   Queue<string*> queue1;
   Queue<string*> queue2;
   queue1 = queue2;

   queue.push("Victor");
   queue.push("Amanda");
   queue.push("Sofia");
   queue.push("Jose");
   queue.push("Manuel");

   cout << "Mostrando el elmento del frente: " << queue.front() << endl;
   cout << "Mostrando el elmento de atras: " << queue.back() << endl;

   queue.pop();
   queue.pop();

   cout << endl;
   cout << "Mostrando el elmento del frente: " << queue.front() << endl;
   cout << "Mostrando el elmento de atras: " << queue.back() << endl;

   queue.print();

   return 0;
}
```

Tingo Tingo Tango

main.cpp

```
#include <iostream>
#include "List.cpp"
#include "Queue.cpp"

using namespace std;
```

```
string tingoTango(List<string> &listaNombres, int N){
   Queue<string> cola;
   for(int i=0; i < listaNombres.size(); i++){
       cola.push(listaNombres.get(i));
   }
   while(cola.size() > 1 ) {
       for (int i = 0; i < N; i++) {
           string elemento = cola.front();
           cola.pop();
           cola.push(elemento);
       }
       cola.pop();
   }
   return cola.front();
}

int main() {
   List<string> lista;
   lista.insert(0, "victor");
   lista.insert(1, "manuel");
   lista.insert(2, "oscar");
   lista.insert(3, "amanda");
   lista.insert(4, "rosa");
   lista.insert(5, "juan");

   cout << endl << tingoTango(lista, 7) << endl;
   return 0;
}
```

Categorización

main.cpp

```
#include <iostream>
#include "List.cpp"
#include "Queue.cpp"
#include <ctime>

using namespace std;

int main() {
   Queue<int> cola1, cola2, cola3, cola4, cola5;
   int ingreso;
   srand(time(0));
   for(int i=0; i<50; i++){
       ingreso = rand() % 50 + 1;
       switch (ingreso / 10) {
           case 0:
               cola1.push(ingreso);
               break;
           case 1:
               cola2.push(ingreso);
               break;
           case 2:
               cola3.push(ingreso);
               break;
           case 3:
```

```
                cola4.push(ingreso);
                break;
            case 4:
                cola5.push(ingreso);
                break;
        }
    }

    cout << "Categoria 1 (entre 0 y 9): ";
    cola1.print();
    cout << endl;

    cout << "Categoria 2 (entre 10 y 19): ";
    cola2.print();
    cout << endl;

    cout << "Categoria 3 (entre 20 y 29): ";
    cola3.print();
    cout << endl;

    cout << "Categoria 4 (entre 30 y 39): ";
    cola4.print();
    cout << endl;

    cout << "Categoria 5 (entre 40 y 49): ";
    cola5.print();
    cout << endl;
    return 0;
}
```