Estructuras de Datos y Librerias

Edson Eddy Lecoña Zarate

Facultad de Ciencias Puras y Naturales
Universidad Mayor de San Andrés

29 de agosto de 2019

Contenido

Tipos de datos C++

Tipos de datos C++

Type	Bytes	Min value	Max value
bool	1	0	1
char	1	-128	127
short	2	-32768	32767
int	4	-2148364748	2147483647
long long	8	-9223372036854775808	9223372036854775807

Type	Bytes	Min value	Max value
unsigned char	1	0	255
unsigned short	2	0	65535
unsigned int	4	0	4294967295
unsigned long long	8	0	18446744073709551615

Type	Bytes	Min value	Max value	Precision
float	4	$\approx -3{,}4 \times 10^{38}$	$\approx 3{,}4 \times 10^{38}$	$\approx$ 7 digits
double	8	$\approx -1{,}7 \times 10^{308}$	$\approx 1{,}7 \times 10^{308}$	$\approx$ 14 digits
long double	16	$\approx -1{,}1 \times \times 10^{4932}$	$\approx 1{,}1 \times 10^{4932}$	$\approx$ 18 digits

Entrada/Salida C++

Entrada/Salida C++

La librería **iostream** es un componente de la biblioteca estándar de C++ que es utilizado en operaciones de entrada/salida.

```
#include <iostream>
```

Entrada
El comando **cin** es para la introducción de datos por teclado, este comando esta seguido por el operador $>>$, a continuación la variable.

```
TipoDato nameVar;
cin>>nameVar;
```

Salida
El comando **cout** es para la impresión de datos por consola, este comando esta seguido por el operador $<<$, a continuación la variable.

```
TipoDato nameVar;
cout<<nameVar;
```

El comando **endl** indica salto de linea.

```
TipoDato nameVar;
cout<<nameVar<<endl;
```

Ejemplo E/S

```
1 #include <iostream>      //importando libreria E/S
2 using namespace std;
3 int main(){
4     int n;               //inicializando variable
5     cin >> n;              //lectura de la variable
6     cout << n << endl;       //impresion de la variable
7     return 0;
8 }
9
```

STL C++

STL C++

La STL (La Standard Template Library) es una colección de estructuras de datos y algoritmos de uso común. La STL se podría dividir en tres grandes partes: Contenedores (plantillas de estructuras de datos populares), iteradores y algoritmos.

Los **Contenedores** de la STL son estructuras de datos capaces de contener casi cualquier tipo de objeto.

Generalizando se tiene:

```
TipoEstructura <TipoDato> name;
```

Vector

vector<tipoDato> nombre;

Características:

- Es una estructura de datos dinámica lineal.
- Es un contenedor secuencial que representa un Arreglo.

Algunos métodos:

push_back(dato)	Insertar un dato al final del Vector.
size()	Retorna la longitud del Vector.
empty()	Retorna true si el Vector está vacío, caso contrario false.
clear()	Elimina todos los datos del Vector.

Vector (Ejemplo)

```cpp
#include <iostream> //importando libreria E/S
#include <vector>   //importando libreria vector
using namespace std;
int main(){
    int n, x;
    cin >> n;
    vector <int> V; // vector<int> V(n) vector estatico
    for (int i = 0; i < n; i++){    //lectura de datos
        cin >> x;
        V.push_back(x); //insertando elemento al vector
    }
    for (int i = 0; i < n; i++){
        cout << V[i] << " " << endl;    //impresion de datos
    }
}

```

Stack (Pila)

stack <tipoDato> nombre;

Características:

- Es una estructura de datos dinámica lineal.
- Sigue el principio de LIFO (Last In First Out).

Algunos métodos:

push(dato)	Introduce un dato en la Pila.
top()	Accede al último elemento de la Pila.
pop()	Elimina el último elemento de la Pila.
size()	Retorna el tamaño de la Pila.
empty()	Retorna true si la Pila está vacía, caso contrario false.

Stack (Ejemplo)

```cpp
#include <iostream> //importando libreria E/S
#include <stack>    //importando libreria stack
using namespace std;
int main(){
    int n, x; cin >> n;
    stack <int> S;  //inicializando pila
    for (int i = 0; i < n; ++i){
        cin >> x; S.push(x); //insertando elemento en la pila
    }
    while(!S.empty()){
        x = S.top();    //extrayendo un elemento de la pila
        S.pop();    //eliminando un elemento de la pila
        cout << x << endl;  //impresion del elemento
    }
    return 0;
}

```

Queue (Cola)

queue <tipoDato> nombre;

Características:

- Es una estructura de datos dinámica lineal.
- Sigue el principio de FIFO (First In First Out).

Algunos métodos:

push(dato)	Introduce un dato en la Cola.
front()	Accede al primer elemento de la Cola.
pop()	Elimina el primer elemento de la Cola.
size()	Retorna el tamaño de la Cola.
empty()	Retorna true si la Cola está vacía, caso contrario false.

Queue (Ejemplo)

```cpp
#include <iostream> //importando libreria E/S
#include <queue>    //importando libreria queue
using namespace std;
int main(){
    int n, x; cin >> n;
    queue <int> S;  //inicializando Cola
    for (int i = 0; i < n; ++i){
        cin >> x; S.push(x);    //insertando elemento en la cola
    }
    while(!S.empty()){
        x = S.front();  //extrayendo elemento de la cola
        S.pop();    //eliminando elemento de la cola
        cout << x << endl;  //mostrando elemento de la cola
    }
    return 0;
}

```

Deque (Bicola)

deque <tipoDato> nombre;

Características:

- Es una estructura de datos dinámica lineal.
- Es una variacion de Pila y Cola.

Algunos métodos:

push_back(dato)	Introduce por detrás un dato a la bicola.
push_front(dato)	Introduce por adelante un dato a la bicola.
front()	Accede al primer elemento de la bicola
back()	Accede al último elemento de la bicola
pop_back()	Elimina el último dato de la bicola.
pop_front()	Elimina el primer dato de la bicola.
size()	Retorna el tamaño de la bicola.
empty()	Retorna true si la bicola está vacía, caso contrario false.

Deque (Ejemplo)

```cpp
#include <iostream> //importando libreria E/S
#include <deque>    //importando libreria deque
using namespace std;
int main(){
    int n, x; cin >> n;
    deque <int> S;  //inicializando la cola
    for (int i = 0; i < n; ++i){
        cin>>x; S.push_front(x);    //insertando elemento al comienzo de la cola
    }
    while(!S.empty()){
        x = S.front();  //extrayendo elemento de la cola
        S.pop_front();  //eliminando elemento de la cola
        cout << x << endl;  //mostrando elemento de la cola
    }
    return 0;
}

```

Priority Queue (Cola de Prioridad)

priority_queue <tipoDato> nombre;

Características:

- Es una estructura de datos dinámica no lineal.
- Es una variacion de Cola.
- Los elementos ingresados se ordenan automaticamente (Segun Prioridad).

Algunos métodos:

push(dato)	Introduce un dato en la Cola.
top()	Accede al primer elemento de la Cola.
pop()	Elimina el primer elemento de la Cola.
size()	Retorna el tamaño de la Cola.
empty()	Retorna true si la Cola está vacía, caso contrario false.

Priority Queue (Ejemplo)

```cpp
#include <iostream> //importando libreria E/S
#include <queue>    //importando libreria queue
using namespace std;
int main(){
    int n, x; cin >> n;
    priority_queue <int> S; //inicializando cola
    for (int i = 0; i < n; ++i){
        cin >> x; S.push(x);    //insertando elemento en la cola
    }
    while(!S.empty()){
        x = S.top();    //extrayendo elemento de la cola
        S.pop();    //eliminando elemento de la cola
        cout << x << endl;  //mostrando elemento de la cola
    }
    return 0;
}

```

Set (Conjunto)

set <tipoDato> nombre;

Características:

- Es una estructura de datos dinámica no lineal.
- No admite elementos repetidos (elementos únicos).
- Sus funciones principales Búsqueda/Eliminacion/Insercion.
- Los elementos ingresados se ordenan automáticamente.
- Solo se puede acceder a sus elementos mediante el uso de iteradores.

Algunos métodos:

insert(dato)	Introduce un dato en el Set.
count()	Consulta si un elemento se encuentra en el Set.
erase(dato)	Encuentra un dato y lo elimina.
clear()	Limpia el contenido del Set.
size()	Retorna el tamaño del Set.
empty()	Retorna true si el Set está vacío, caso contrario false.

Set (Ejemplo)

```cpp
#include <iostream> //importando libreria E/S
#include <set>  //importando libreria set
using namespace std;
int main(){
    int n, x; cin >> n;
    set <int> S;    //inicializando set
    for (int i = 0; i < n; ++i){
        cin >> x; S.insert(x);  //insertando elemento al set
    }
    cout<< S.count(1) << endl;  //Consulta si 1 se encuentra en el set
    cout<< S.count(3) << endl;  //Consulta si 3 se encuentra en el set
    cout<< S.count(5) << endl;  //Consulta si 5 se encuentra en el set
    return 0;
}

```

Map (Mapa)

map <tipoDato, tipoDato> nombre;

Características:

- Es una estructura de datos dinámica no lineal.
- Los elementos se almacenan en la estructura según Llave-Valor.
- No admite elementos repetidos (llave única).
- Sus funciones principales Búsqueda/Eliminación/Inserción.
- Los elementos ingresados se ordenan automáticamente (Según llave).

Algunos métodos:

[llave]	Accede un elementos del Map mediante su llave.
erase(dato)	Elimina el dato buscado si se encuentra en el Map.
size()	Retorna el tamaño del Map.
empty()	Retorna true si el Map está vacío, caso contrario false.

Map (Ejemplo)

```cpp
#include <iostream> //importando libreria E/S
#include <map>  //importando libreria map
using namespace std;
int main(){
    int n, x; string st;
    cin >> n;
    map <string, int> S;    //inicializando map, llave=string, valor=int
    for (int i = 0; i < n; ++i){
        cin >> x >> st;
        S[st] = x;  //asignando valor x a la llave st
    }
    cout<< S["abc"] << endl;    //mostrando lo que contiene la llave abc
    cout<< S["ab"] << endl; //mostrando lo que contiene la llave ab
    cout<< S["a"] << endl;  //mostrando lo que contiene la llave a
    return 0;
}

```

Iterator

tipoEstructura <TipoDato> :: iterator nombre;

Un **iterator** es un objeto que se mueve a través de un contenedor de otros objetos y selecciona a uno de ellos cada vez, sin proporcionar un acceso directo a la implementación del contenedor.

Los iteradores proporcionan una forma estándar de acceder a los elementos.

métodos que es utilizaran:

.begin()	Retorna el iterador del primer elemento, si el está vació retorna .end().
.end()	Retorna el iterador del elemento siguiente al último elemento

Iterator (Ejemplo)

```cpp
#include <bits/stdc++.h>
using namespace std;
int main(){
    set <int> S;     //inicializando Set
    S.insert(12);    //insertando elementos
    S.insert(0);
    S.insert(5);
    set <int> :: iterator it;    //inicializando iterador
    for (it = S.begin(); it != S.end(); it++){
        cout<<*it<<endl;     //mostrando elemento que apunta el iterador
    }
    return 0;
}
```

Ejercicios Propuestos

Ejercicios Propuestos

Lista de Problemas	
jv.umsa.bo	uva.onlinejudge.org
Redundancia	The Departament of Redundancy
Primer Diccionario	Andy's First Dictionary
al revés	
Bob y los parentesis	Parentheses Balance
Letras repetidas	