[Γα=Ω5]

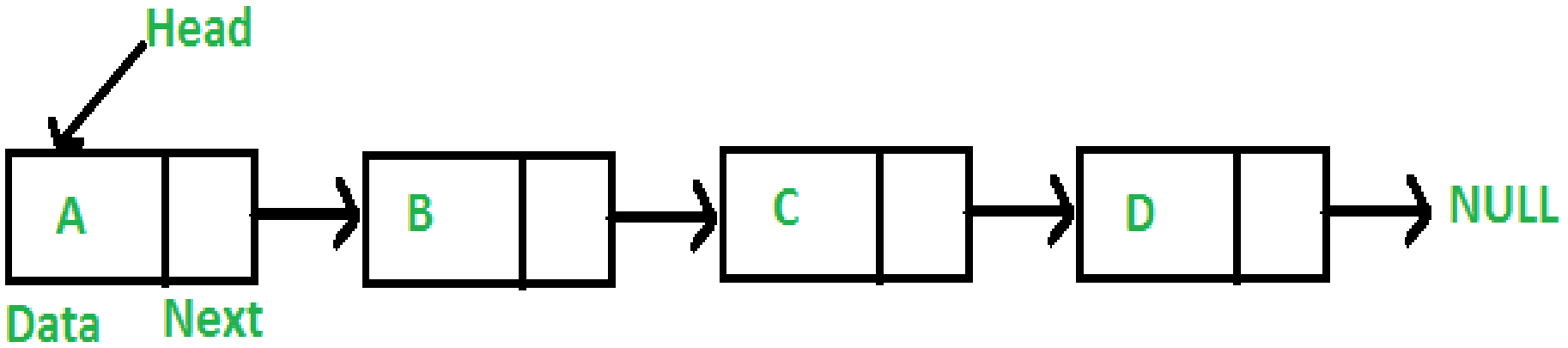
STL Containers

Por Alan Martínez

[Γα=Ω5]

STL containers (Estructuras de Datos Lineales)

Son estructuras de datos que forman parte de la Biblioteca de Plantillas Estándar en C++ (STL). Están diseñadas para almacenar y gestionar colecciones de elementos de manera eficiente e incluiyen funciones dentro del mismo tipo de contenedor con el operador punto .



Vectores

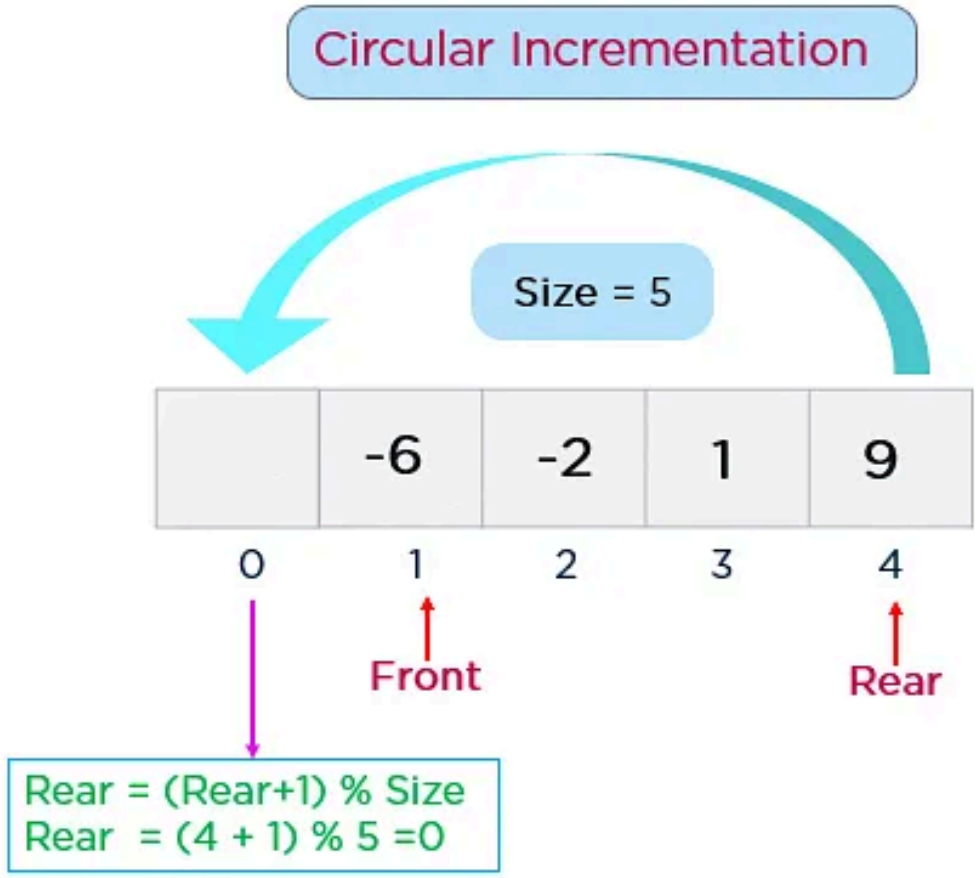
Los vectores son contenedores que almacenan elementos de un mismo tipo en un bloque de memoria **contiguo**. Son similares a los arreglos, pero ofrecen una mayor flexibilidad en cuanto a tamaño y operaciones.

Ejemplos:

```cpp
#include <vector>
vector<int> vec = {1, 2, 3, 4, 5};
vector<int> vec{1, 2, 3, 4, 5}; // Similar al anterior
vector<int> vec (n);// vector de tamaño n
vector<int> vec (n,0);// vector de tamaño n y llenado con puros ceros
vector<vector<int>> matriz;
vec.empty(); //true (1) si esta vacia, false (0) sino
vec.push_back(6); // Agregar un elemento al final
vec.pb(6); // push_back acortado de la plantilla
vec.emplace_back(valor); // como push_back, pero más veloz
vec.size(); // tamaño
vec.empty(); // limpia los contenidos del vector
int a = vec.pop_back(); //elimina el elemento del final
```

Arrays Circulares

Un array circular es una estructura de datos en la que el último elemento está conectado al primero, formando un círculo. Se pueden implementar usando un vector normal y realizando operaciones de módulo para simular la circularidad.



Distancias de un array circular

Distancias:

```
vector <int> vec = {0, 1, 2, 3, 4, 5};
int n = vec.size();
const int idxI = 3, idxF = 1; //indices inicial y final
int distancia_izquierda = abs(idxI-idxF);
int distancia_derecha = n - distancia_izquierda;
int distancia_minima = min(distancia_izquierda,distancia_derecha);
```

Recorridos de un array circular

siendo `n` el tamaño del vector.

Recorrido por la izquierda:

```
for (int i = idxI; i != idxF; i = (i - 1 + n) % n) {
    cout << array[i] << " ";
}
```

Recorrido por la derecha:

```
for (int i = idxI; i != idxF; i = (i + 1 + n) % n) {
    cout << array[i] << " ";
}
```

Bitsets

Los bitsets son contenedores especializados para almacenar bits individuales permitiendo hacer operaciones bit a bit (bitwise). Es un remplazo útil para vectores boolenos también conocidos como vectores de banderas.

Ejemplo:

```
#include <bitset>  // Para bitset
bitset<N> boolarr; // Donde N es el número de bits, por defecto se inicializan todos en 0
    b.set(1);        // Establece el bit en la posición 1
    b.flip(3);       // Cambia el bit en la posición 3
    cout << "Bits: " << b << endl;
    cout << "número de 1s: " << b.count() << endl;
    cout << "Tamaño: " << b.size() << endl;
    cout << "Bit en posición [1]: " << b.test(1) << endl;
```

Strings

los strings en c son solo arrays de `char`, pero en c++ son un contenedor de la Biblioteca Estándar (STL). A diferencia de los strings de c, estos permiten realizar operaciónes de asignación o evaluación booleana de una forma más fácil.

Ejemplo:

```
#include <string>
string str = "Alan";
str += " Martínez";
str.size();
str.lenght(); // lo mismo que .size(), pero exclusivo para strings
if (str == "Alan Martínez") cout << "Yey!";
if (str.compare("Alan Martínez")) cout << "Yey!";
str.empty(); //true (1) si esta vacia, false (0) sino
str = "Hello, world!";
// Reemplaza "world" (empezando en la posición 7) con "C++"
str.replace(7, 5, "C++");// 5 es la longitud de "world"
cout << str;
```

Getline

Esta es una función incorporada que acepta entradas de un solo carácter o múltiples caracteres, a diferencia del cin, acepta espacios.

```
string sentence;
//cin.getline(sentence, 100); // ocupa tamaño, por lo que es inutil en este caso
getline(cin,sentence) // captura una sentencia completa
cin.ignore();//para leer más datos después
//cin.ignore(numeric_limits<streamsize>::max(), '\n'); // por si no funciona lo de arriba
cout << sentence <<"/n";
```

Streamstrings

En C++, stringstream es una clase de la biblioteca estándar (STL) que permite realizar operaciones concatenacion de entrada y salida en un flujo de datos basado en strings.

```
#include <string>
#include <sstream>

string nombre = "Alan";
stringstream ss;
ss << "Hola, " << nombre << "!";
```

substrings y strings circulares

Un substring es un string contenido dentro de un string padre, por ejemplo la palabra `"locomotora"` contiene el substring `"loco"`.

El siguiente código genera dos vectores `vIZQ` y `vDER` que contienen todas las rotaciones posibles de la cadena `s`, pero en diferentes órdenes. `vIZQ` contiene las rotaciones de la cadena en el orden que resulta al mover la parte inicial hacia el final, mientras que `vDER` contiene las rotaciones de la cadena en el orden que resulta al mover la parte final hacia el inicio.

```
string s; cin >> s;
vector<string> vIZQ(s.size()); vector<string> vDER(s.size());
for(int i=0;i<s.size();i++){
    vIZQ[i] = s.substr(i,s.size()-i) + s.substr(0,i);
    vDER[i] = s.substr(s.size()-i,i) + s.substr(0,s.size()-i);
}
```

funcionamiento del código anterior

Supongamos que la cadena `s` es `"abcd"`.

- Para `i = 0`:
  - `vIZQ[0] = s.substr(0, 4) + s.substr(0, 0) = "abcd" + "" = "abcd"`
  - `vDER[0] = s.substr(4 - 0, 0) + s.substr(0, 4) = "" + "abcd" = "abcd"`
- Para `i = 1`:
  - `vIZQ[1] = s.substr(1, 3) + s.substr(0, 1) = "bcd" + "a" = "bcda"`
  - `vDER[1] = s.substr(4 - 1, 1) + s.substr(0, 3) = "d" + "abc" = "dabc"`
- Para `i = 2`:
  - `vIZQ[2] = s.substr(2, 2) + s.substr(0, 2) = "cd" + "ab" = "cdab"`
  - `vDER[2] = s.substr(4 - 2, 2) + s.substr(0, 2) = "cd" + "ab" = "cdab"`
- Para `i = 3`:
  - `vIZQ[3] = s.substr(3, 1) + s.substr(0, 3) = "d" + "abc" = "dabc"`
  - `vDER[3] = s.substr(4 - 3, 3) + s.substr(0, 1) = "abc" + "d" = "abcd"`

Palíndromos

Un string se llama palíndromo si el reverso del string es igual al original.

```
bool esPalindromo(const string& str) {
    string strLimpio;

    for (char ch : str) { // Eliminar caracteres no alfanuméricos y convertir a minúsculas
        if (isalnum(ch)) {
            strLimpio += tolower(ch);
        }
    }

    string strReverso = strLimpio;
    reverse(strReverso.begin(), strReverso.end());

    return strLimpio == strReverso;
}
```

Pair y Tuple

En C++, tanto `pair` como `tuple` son estructuras de datos útiles para agrupar múltiples valores en una sola unidad. Cada una tiene sus propias características y usos específicos.

`pair` es una estructura de datos que almacena dos elementos de posiblemente diferentes tipos. Es parte de la biblioteca estándar de C++ y se define en el header `<utility>`.

```
pair<string, int> p1;
auto p2 = make_pair("Hello", 42);
p1.first = "Hello";
p1.second  = 42;
cout << p2.F << p2.S; // con el define de la plantilla

if (p1 == p2) cout << "Son iguales";
```

la tupla o `tuple` es una estructura de datos que puede almacenar un número arbitrario de elementos de diferentes tipos:

```
#include <utility>  // Para std::tie
tuple <int,double,string> t1;
auto t2 = make_tuple(10, 20.5, "World");


int entero = get<0>(t1);
double decimal = get<1>(t1);
string texto = get<2>(t1);

tie(entero, decimal, texto) = t2; // obtener valores con tie
```

Algoritmos útiles del STL

```
int minimo = min(a,b);
minimo = min({a,b,c,d})
int maximo = max(a,b);
maximo = max({a,b,c,d})
auto [minVal, maxVal] = minmax(a, b);
cout << minVal << maxVal;

//sumatoria notación sigma ( Σ )
int i = 0; // inicio de la suma
int sum = accumulate(v.begin(), v.end(), i);
// multiplicatoria, notación pi ( Π )
int i = 1; // valor multiplicativo inicial
int product = accumulate(v.begin(), v.end(), i, multiplies<int>());
// acumulación personalizada con lamda
int result = std::accumulate(v.begin(), v.end(),i, [](int a, int b) {
  return a+b;//operación
});
```

```
merge(v1.begin(), v1.end(), v2.begin(), v2.end(), back_inserter(vecR));
// en este segundo caso declarar a vecR con la suma de ambos tamaños
merge(vec1.begin(), vec1.end(), vec2.begin(), vec2.end(), vecR.begin());

// Llena todos los elementos del vector con el valor `value`
fill(v.begin(), v.end(), value);

// Llena el vector con valores secuenciales empezando en `start`
iota(vec.begin(), vec.end(),start);
```

Problemas

- **71A** Way Too Long Words
- **731A** Night at the Museum

Referencias

- Gaurav. (2024). *How to Implement a Circular Buffer Using std::vector?*. Recuperado de https://www.geeksforgeeks.org/implement-circular-buffer-using-std-vector-in-cpp/
- GeeksforGeeks. (2023). *Program to print all substrings of a given string*. Recuperado de https://www.geeksforgeeks.org/program-print-substrings-given-string/
- GeeksforGeeks. (2024). *C++ STL Cheat Sheet*. Recuperado de https://www.geeksforgeeks.org/cpp-stl-cheat-sheet/
- GeeksforGeeks. (2024). *Substring in C++*. Recuperado de https://www.geeksforgeeks.org/substring-in-cpp/
- GeeksforGeeks. (2024). *Vector in C++ STL*. Recuperado de https://www.geeksforgeeks.org/vector-in-cpp-stl/
- Imam, M. (2023). *Pair in C++ Standard Template Library (STL)*. Recuperado de https://www.geeksforgeeks.org/pair-in-cpp-stl/

- Kumar, H. (2024). *String | Palindrome*. Recuperado de https://www.geeksforgeeks.org/string-palindrome/
- mrityuanjay8vae. (2024). *String Guide for Competitive Programming*. Recuperado de https://www.geeksforgeeks.org/strings-for-competitive-programming/
- Pal, S. (2019). *What are some tips for code (C++) optimization to get a better time in competitive programming?*. Recuperado de https://www.quora.com/What-are-some-tips-for-code-C-optimization-to-get-a-better-time-in-competitive-programming?lang=www
- Singh, M. (2023). *Tuples in C++*. Recuperado de https://www.geeksforgeeks.org/tuples-in-c/
- Striver. (2024). *Circular array*. Recuperado de https://www.geeksforgeeks.org/circular-array/