TD3: Algoritmos y Estructuras de Datos

Prof. Gervasio Pérez
Prof. Agustín Martinez Suñé

Primer Semestre de 2023

Clase teórica 2: C++

Continuamos con C++

- `std`: la biblioteca estándar de C++
- Tipos `string`, `vector`, `set`, `map`.
- Uso de iteradores.
- Entrada y salida de archivos.

Biblioteca estándar de C++

- Provee funcionalidades comunes que facilitan la escritura de nuestros programas.
- Está programada enteramente en C++.
- Entre otras cosas, provee algoritmos y estructuras de datos conocidas que podemos utilizar.
- La referencia oficial del lenguage y la biblioteca es el estándar ISO 14882:2020(E) - Programming Language C++.
- Nosotros usaremos la documentación desarrollada y mantenida por la comunidad en `https://cppreference.com`.

Biblioteca estándar de C++

- Cada funcionalidad de la biblioteca estándar se provee a través de algún header estándar.

```
1  #include<string>
2  #include<vector>
```

- Todas las funcionalidades de la biblioteca estándar se define en el espacio de nombres std.

```
1  std::string mi_cadena = "cadena";
2  std::vector<int> mi_vector = {1,4,6,9};
```

- Se puede declarar std como espacio de nombres por defecto para no tener que escribirlo explícitamente.

```
1  #include<string>
2  using namespace std;
3  string mi_cadena = "cadena";
```

Por simplicidad, a veces omitiremos el uso explícito de std.

Tipo string

La clase string permite la manipulación de cadenas de **char**.

- Podemos inicializar una variable utilizando comillas dobles.

```
1   string universidad = "universidad Torcuato Di Tella";
```

- Leer y escribir en una posición determinada.

```
1   char letra = universidad[5];
2   universidad[0] = 'U';
```

- Consultar la cantidad de caracteres de la cadena.

```
1   int cantidad = universidad.size();
2   // También se puede utilizar universidad.length()
```

- Acceder al primer y al último caracter.

```
1   char primero = universidad.front();
2   char ultimo = universidad.back();
```

- Cuidado: la manipulación a nivel de **char** no funciona bien con caracteres unicode.

Tipo `string`

- Evaluar si la cadena está vacía.

```
1   bool esta_vacia = universidad.empty();
```

- Obtener una subcadena.

```
1   string subcadena = universidad.substr(12, 8);
```

- Modificar la cadena agregando otra cadena al final.

```
1   universidad.append(", Argentina.");
2   // "Universidad Torcuato Di Tella, Argentina."
```

- Reemplazar una subcadena por otra cadena.

```
1   universidad.replace(12, 8, "Guido");
2   // "Universidad Guido Di Tella, Argentina."
```

- Pueden encontrar más funcionalidades en https://es.cppreference.com/w/cpp/string/basic_string

Tipo vector

La clase `vector` es el contenedor más usado de C++, con funcionalidades similares al tipo `list` de Python.

- Es una secuencia de elementos del mismo tipo.

```
vector<int> secuencia_de_enteros;
vector<float> secuencia_de_float;
vector<string> secuencia_de_strings;
```

- Se puede inicializar con una lista de elementos.

```
vector<int> numeros = {5, 7, 29, 10, 23};
```

- Leer y escribir el elemento en cada posición.

```
int n = numeros[3];
numeros[1] = 8; // {5, 8, 29, 10, 23}
```

- Acceder al primer y último elemento con `.front()` y `.back()`.

Tipo vector

- Consultar la cantidad de elementos con `.size()`.
- Evaluar si está vacío con `.empty()`.
- Añadir un elemento al final.

```
1   numeros.push_back(25);
2   // {5, 8, 29, 10, 23, 25}
```

- Eliminar el último elemento.

```
1   numeros.pop_back();
2   // {5, 8, 29, 10, 23}
```

- Pueden encontrar más funcionalidades en `https://es.cppreference.com/w/cpp/container/vector`

Tipo set

La clase set provee un contenedor de elementos del mismo tipo donde no hay repeticiones y ni un orden específico. Similar al tipo `set` de Python.

- Se puede incluir con el header set:

```
#include <set>
```

- Inicializar con una lista de elementos:

```
set<string> nombres = {"Alice", "Bob", "Carol"};
set<int> conjunto_numeros = {1, 7, -1, 10, 9};
```

- Consultar la cantidad de elementos con `.size()` y evaluar si está vacío con `.empty()`.
- Consultar si un elemento pertenece al conjunto o no:

```
int a = nombres.count("Alice"); // a == 1
int b = nombres.count("Charlie"); // b == 0
```

Tipo set

- Insertar un nuevo elemento:

```
1   nombres.insert("Charlie");
2   // {"Alice", "Bob", "Carol", "Charlie"}
3   nombres.insert("Alice");
4   // {"Alice", "Bob", "Carol", "Charlie"}
```

- Eliminar un elemento del conjunto:

```
1   nombres.erase("Alice");
2   // {"Bob", "Carol", "Charlie"}
3   nombres.erase("Beta");
4   // {"Bob", "Carol", "Charlie"}
```

- Pueden encontrar más funcionalidades en https://es.cppreference.com/w/cpp/container/set

Tipo map

La clase map provee un contenedor asociativo en el que se tienen claves de cierto tipo asociadas a valores de (potencialmente) otro tipo. Similar al tipo `dict` de Python.

- Se puede incluir con el header map:

```
#include <map>
```

- Inicializar con una lista de elementos:

```
map<string, int> stock =
  { {"vasos", 4}, {"tenedores", 4}, {"cuchillos", 3} };
map<int, string> numero_a_palabra =
  { {1, "Uno"}, {5, "Cinco"}, {2, "Dos"} };
```

- Acceder y escribir el valor asociado a una clave determinada.

```
int cant_vasos = stock["vasos"];
stock["cuchillos"] = stock["cuchillos"] + 1;
```

Tipo map

- Insertar una clave nueva y asociarla a un valor:

```
1   stock["tazas"] = 3;
```

- Consultar la cantidad de elementos con `.size()` y evaluar si está vacío con `.empty()`.
- Consultar si una clave está definida en el contenedor o no:

```
1   int a = stock.count("tenedores"); // a == 1
2   int b = nombres.count("cucharas"); // b == 0
```

- Pueden encontrar más funcionalidades en `https://es.cppreference.com/w/cpp/container/map`

Definición de **enum class**

Permite definir un nuevo tipo de datos a partir de una enumeración de constantes. Similar a la clase `Enum` de Python.

```
1 ▶  enum class Estacion = {Otonio, Invierno, Primavera, Verano};
```

- Crea un espacio de nombres para acceder a cada constante:

```
1    Estacion actual = Estacion::Verano;
```

- Permite comparar por igualdad dos instancias del tipo enumerado:

```
1    if(actual == Estacion::Invierno){
2      cout << "¡Qué frío!" << endl;
3    }
```

- Pueden encontrar más información en `https://es.cppreference.com/w/cpp/language/enum`

Definición de **struct**

Permite definir una estructura de datos, es decir, un tipo de datos compuesto de varios campos de distintos tipos.

- Se debe declarar el nombre y el tipo de cada campo:

```
1  struct MatrizEnergetica {
2    int produccion_total_anual;
3    float porcentaje_fosil;
4    float porcentaje_nuclear;
5    float porcentaje_renovables;
6    string fuente_informacion;
7  }
```

- Se puede inicializar con los valores de cada campo:

```
1  MatrizEnergetica matriz_argentina =
2    {135, 84.16, 3.01, 12.83, "desconocida"};
3  MatrizEnergetica matriz_alemania =
4    {581, 77.42, 4.83, 17.75, "OurWorldInData.org"};
```

Definición de **struct**

- Se puede acceder a y modificar el valor de cada campo:

```
1   matriz_argentina.fuente_informacion = "OurWorldInData.org";
```

- Esta construcción se hereda del lenguaje de programación C. La utilizaremos únicamente para definir tipos compuestos como este que, a diferencia de las clases, no proveen métodos o funciones.
- Pueden encontrar más información en `https://en.cppreference.com/w/c/language/struct`.

Uso de iteradores

- Un iterador es un objeto que permite recorrer los elementos de un contenedor.
- La mayoría de los contenedores de la biblioteca estándar proveen iteradores. En particular `vector`, `set`, `map` los proveen.
- Son especialmente útiles para recorrer estructuras que, a diferencia de `vector`, no proveen acceso arbitrario estilo `v[i]`.

Uso de iteradores

- ¿Cómo recorríamos todos los elementos de un vector?

```
vector<int> v = {5, 8, 20, 17, 0};

for(int i = 0; i < v.size(); i++){
    cout << v[i] << endl;
}
```

- Los iteradores generalizan esa forma de recorrer un contenedor.

```
for(vector<int>::iterator it = v.begin(); it != v.end(); it++){
    cout << *it << endl;
}
```

- El **for** "a la Python" que itera los elementos de un contenedor es en realidad una forma simplificada de usar iteradores.

```
for(int e : v){
    cout << e << endl;
}
```

Uso de iteradores

Podemos recorrer los elementos de un set utilizando iteradores. Dado un iterador it de set, el valor iterado *it **siempre será const**: no se puede cambiar un valor almacenado en el set por medio del iterador.

```
1  set<int> s = {5, 8, 20, 17, 0};
2
3  for(set<int>::iterator it = s.begin(); it != s.end(); it++){
4      cout << *it << endl;
5  }
6
7  cout << endl;
8
9  for(int e : s){
10     cout << e << endl;
11 }
```

Uso de iteradores

Podemos recorrer los elementos de un map utilizando iteradores. Dado un iterador it de map, el componente it->first **siempre será const**: no se puede cambiar una clave por medio del iterador.

```
map<string, int> m =
  {{"a", 5}, {"b", 8}, {"c", 20}, {"d", 17}, {"e", 0}};

for(map<string, int>::iterator it = m.begin(); it != m.end(); it++){
    cout << it->first << " => " << it->second << endl;
}
```

```
for(pair<string, int> e : m){
    cout << e.first << " => " << e.second << endl;
}
```

Uso de iteradores

En los casos en que los nombres de los tipos se vuelven muy largos y tediosos podemos usar (¡CON CUIDADO!) la keyword **auto** para que el compilador infiera el tipo correspondiente.

```
map<string, int> m =
  {{"a", 5}, {"b", 8}, {"c", 20}, {"d", 17}, {"e", 0}};

for(auto it = m.begin(); it != m.end(); it++){
    cout << it->first << " => " << it->second << endl;
}
```

```
for(auto e : m){
    cout << e.first << " => " << e.second << endl;
}
```

Uso de iteradores

- Los iteradores además nos permiten utilizar algoritmos de la biblioteca estándar que están preparados para ejecutar sobre cualquier contenedor que provea iteradores.
- Por ejemplo `std::count` definido en el header `<algorithm>`:

```
1  #include <algorithm>
2  #include <vector>
3  #include <iostream>
4
5  using namespace std;
6
7  int main(){
8      vector<int> v = {6, 38, 40 ,54, 1, 40};
9      int cantidad_de_cuarentas = count(v.begin(), v.end(), 40);
10 }
```

Entrada/salida de archivos

- La biblioteca estándar provee funcionalidades para leer y escribir archivos.
- Conceptualmente es muy similar a la entrada/salida en la consola.
- Debemos incluir el encabezado

```
1  #include <fstream>
```

- Podemos inicializar un *flujo* de lectura de un archivo:

```
1  ifstream infile ("path/to/file");
```

- El objeto llamado `infile` soporta todas las operaciones de `cin`.
- Podemos inicializar un *flujo* de escritura de un archivo:

```
1  ofstream outfile ("path/to/file");
```

- El objeto llamado `outfile` soporta todas las operaciones de `cout`.
- Debemos cerrar cada archivo utilizando el método `.close()`.

Entrada/salida de archivos

Ejemplo de programa que lee e imprime todas las líneas de un archivo.

```
1  #include <fstream>
2  #include <string>
3  #include <iostream>
4  using namespace std;
5
6  int main(){
7      string filename = "archivo.txt";
8      ifstream infile (filename);
9
10     if (infile.good() == false) {
11         cout << "No se pudo abrir el archivo " << filename << endl;
12             exit (1);
13     }
14
15     string line;
16     while(!infile.eof()){
17         getline(infile, line);
18         cout << line << endl;
19     }
20
21     infile.close();
22 }
```

Entrada/salida de archivos

Ejemplo de programa que escribe en un archivo.

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

int main(){
    string filename = "archivo.txt";
    ofstream outfile (filename);

    if (outfile.good() == false) {
        cout << "No se pudo abrir el archivo " << filename << endl;
        exit (1);
    }

    string palabra;
    cin >> palabra;
    while(palabra != "fin"){
        outfile << palabra << endl;
        cin >> palabra;
    }

    outfile.close();
}
```

¿Qué vimos hoy?

- Cómo usar la biblioteca estándar de C++
- Aprendimos a usar los principales *contenedores* provistos por la biblioteca estándar: `string`, `vector`, `set`, `map`.
- Aprendimos a recorrer estos contenedores usando *iteradores*.
- Entrada y salida de archivos.
- Con esto ya pueden resolver por completo la Guía 1.