C++ Standard Library (STL)

Sofía Beatriz Pérez
Daniel Agustín Rosso

sperez@iua.edu.ar
drosso@iua.edu.ar

Centro Regional Univesitario Córdoba
Instituto Univeristario Areonáutico

Informática II - Clase número 10 - Ciclo lectivo 2022

Agenda

Disclaimer

Los siguientes slides tienen el objetivo de dar soporte al dictado de la asignatura. De ninguna manera pueden sustituir los apuntes tomados en clases y/o la asistencia a las mismas.
Es importante mencionar que todos este material se encuentra en un proceso de mejora continua.
Si encuentra bugs, errores de ortografía o redacción, por favor repórtelo a sperez@iua.edu.ar y/o drosso@iua.edu.ar. También puede abrir issues en el repositorio de este link: ▸ infoI_IUA_GitLab

Templates: Introducción

Los templates son una de las herramientas más poderosas y sofisticadas de C++. El uso de templates es posible crear funciones y clases genéricas.

El tipo de datos sobre los cuales una función genérica opera se especifica mediante un parámetro. Es decir, que se puede re-utilizar con diferentes tipos de datos sin necesidad de codificar una para cada tipo de datos.

Templates: sintaxis

```
1 template <class Ttype> ret_type func-name (parameters)
2 {
3     cuerpo de la funcion
4 }
```

- Template: palabra reservada de C++
- Type: placeholder para indicar el tipo de dato
- ret_type: tipo de dato retornado por la función
- func_name: nombre de la función
- parameters: parámetros formales de la función

Es importante aclarar el compilador *expande* cada implementación particular en **tiempo de compilación**

Templates: ejemplo I

```
1  #include <iostream>
2  using namespace std;
3
4  template <typename T> T Pmax(T x, T y)
5  {
6      if(x>y)
7      {
8          return(x);
9      }
10     else
11     {
12         return(y);
13     }
14 }
15
```

Templates: ejemplo II

```
16  int main()
17  {
18      cout << Pmax<int>(30, -10) << endl;
19      cout << Pmax<double>(3.1, 7.0)<< endl;
20      cout << Pmax<char>('d', 'a')<< endl;
21      return 0;
22  }
```

Templates: ejemplo I

```
1  #include <iostream>
2  using namespace std;
3
4  template <class T> void swap(T *num1, T *num2)
5  {
6      T temp;
7      temp = *num1;
8      *num1 = *num2;
9      *num2 = temp;
10
11 }
12
13 int main()
14 {
15     int i=10, j=20;
```

Templates: ejemplo II

```
16      double k=50.30, l=90.12;
17
18      cout <<"Original i j "<< i << " " << j<<endl;
19      swap(&i,&j);
20      cout <<"Intercambiado i j "<< i << " " << j<<endl;
21      cout <<"Original k l " << k << " " << l<<endl;
22      swap(&k,&l);
23      cout <<"Intercambiado k l "<< k << " " << l <<endl;
24      return 0;
25  }
```

Introducción a la libreria STL I

La libería estandar de C++ (STL) provee clases, funciones y algoritmos para resolver problemas que suelen ser recurrentes cuando se desarrolla software.

La liberaría STL está basada mayormente en tres items fundamentales:

- Contenedores (containers)
- Algoritmos
- Iteradores

Introducción a la libreria STL II

Containers

Son objetos capaces de almacenar a otros objetos en su interior.
Se clasifican en:

- Secuenciales: almacenan los objetos de forma secuencial permitiendo el acceso a los mismos de forma secuencial y/o aleatoria en función de la naturaleza del contenedor.
- Asociativos: cada objeto almacenado en el contenedor tiene asociada una clave. Mediante la clave los objetos se pueden almacenar en el contenedor, o recuperar del mismo, de forma rápida.

Introducción a la libreria STL III

Algoritmos

Proveen la forma de manipular el contenido de los contenedores. Las capacidades de la libreria STL incluyen:

- Inicializacion
- Ordenamiento
- Búsqueda

La mayoría de los algoritmos opera con un rango de elementos dentro de un contenedor.

Introducción a la libreria STL IV

Iteradores

Los iteradores son objetos que actuan de forma "similar" a los punteros. Proveen al desarrollador la capacidad de iterar dentro de los contendedores de forma similar a la aritmética de punteros. Hay cinco tipo de iteradores:

- Acceso aleatorio
- Acceso bidireccional
- Forward
- Entrada
- Salida

Vectores I

Un vector es una clase genérica que permite almacenar una colección de objetos del mismo tipo. Al igual que con los arreglos ya estudiados, se puede acceder a los elementos del contenedor vector usando un índice y el operador de indexación [] y los elementos ocupan posiciones contiguas en memoria.

```
1 //Vector vacio de enteros
2 vector<int> iv;
3 //Vector de 5 elementos char
4 vector<char>cv(5);
5 //Vector de 5 elementos char inicializados
6 vector<char>cvi(5,"a");
7 //De forma generica
8 vector<T>N;
```

Vectores II

Operaciones básicas con arreglos:

- size(): devuelve el tamaño actual del vector
- begin(): devuelve un iterador que apunta al primer elemento del vector
- end(): devuelve un iterador que apunta al final del vector
- push_back(): agrega un valor al final del arreglo. Si es necesario, el vector es redimensionado automáticamente
- insert(): se puede utilizar para agregar elementos en el medio del arreglo
- erase(): borrado de elementos

Vectores III

```
1  #include <iostream>
2  #include <vector>
3  #include <cctype>
4  using namespace std;
5  int main (void)
6  {
7      int ii;
8      vector<int> v(10);
9      cout<< "Tam de v: "<< v.size()<<endl;
10     //asignando valores
11     for(ii=0;ii<v.size();ii++)
12         v[ii]=ii;
13     //Recorriendo
14     for(ii=0;ii<v.size();ii++)
15         cout<< " "<<v[ii]<<" "<<endl;
```

Vectores IV

```
16      //Agregando al final
17      for(ii=0;ii<5;ii++)
18          v.push_back(10+ii);
19
20      cout<<"Nuevo tam de v: "<<v.size()<<endl;
21      //Recorriendo
22      for(ii=0;ii<v.size();ii++)
23          cout<<" "<<v[ii]<<" "<<endl;
24      return(0);
25  }
```

Vectores utilizando iteradores I

Los elementos de un vector pueden ser accedidos usando el operador [] (ejemplo anterior) o también mediante el uso de iteradores:

```
1  #include <iostream>
2  #include <vector>
3  #include <cctype>
4  using namespace std;
5
6  int main (void)
7  {
8      int ii=0;
9      vector<int> v(10);
10     vector<int>::iterator p;
11
```

Vectores utilizando iteradores II

```
12      p=v.begin();
13      while(p!= v.end())
14      {   *(p)=ii;
15          ii++;
16          p++;
17      }
18      cout<< "Tam de v: "<< v.size()<<endl;
19      p=v.begin();
20      while(p!= v.end())
21      {
22          cout<< " "<< *(p) <<endl;
23          p++;
24      }
25      return(0);
26  }
```

Vectores utilizando iteradores III

Utilizando las funciones insert() y delete():

```
1  #include <iostream>
2  #include <vector>
3  #include <cctype>
4  using namespace std;
5
6  int main (void)
7  {
8      int ii=0;
9      vector<int> v(10);
10     vector<int>::iterator p;
11
12     p=v.begin();
13
14
```

Vectores utilizando iteradores IV

```
15      while(p!= v.end())
16      {   *(p)=ii;
17          ii++;
18          p++;
19      }
20      p=v.begin();
21      cout<< "Tam de v: "<< v.size()<<endl;
22      v.insert(p+3,10);
23      p=v.begin();
24      cout<< "Tam de v: "<< v.size()<<endl;
25      while(p!= v.end())
26      {
27          cout<< " "<< *(p) <<endl;
28          p++;
29      }
```

Vectores utilizando iteradores V

```
30      p=v.begin();
31      v.erase(p+3);
32      cout<< "Tam de v: "<< v.size()<<endl;
33      p=v.begin();
34      while(p!= v.end())
35      {
36          cout<< " "<< *(p) <<endl;
37          p++;
38      }
39      return(0);
40  }
```

Listas I

Las listas son contenedores de secuencias que permiten operaciones de inserción y borrado en cualquier lugar dentro de la secuencia, como así también la iteración en ambas direcciones (listas doblemente enlazadas). La principal diferencia de las listas en comparación con otros contenedores es que carecen de acceso directo a los elementos por su posición.

Operaciones básicas con listas:

- size(): devuelve el tamaño actual de la lista
- begin(): devuelve un iterador que apunta al primer nodo
- end(): devuelve un iterador que apunta al último nodo
- push_back(): agrega un nodo al final

Listas II

- push_front(): agrega un nodo al inicio
- insert(): agrega un nodo en el medio
- erase(): borrado de elementos
- merge(): permite unir dos listas

```
1 #include <iostream>
2 #include <iterator>
3 #include <list>
4  using namespace std;
5
6  void print_lst(list<int> );
7
8
9
```

Listas III

```
10  int main()
11  {
12
13      list<int> l1, l2;
14
15      for (int i = 0; i < 10; ++i)
16      {
17          l1.push_back(i);
18          l2.push_front(i*2);
19      }
20      cout << "l1:"<<endl;
21      print_lst(l1);
22
23      cout << "l2:"<<endl;
24      print_lst(l2);
```

Listas IV

```
25
26      cout << "l1 front: " << l1.front()<<endl;
27      cout << "l1 back: " << l1.back()<<endl;
28      l1.pop_front();
29      cout << "l1:"<<endl;
30      print_lst(l1);
31      cout << "l2:"<<endl;
32      l2.pop_back();
33      print_lst(l2);
34      l1.merge(l2);
35      cout << "l1 merged:"<<endl;
36      print_lst(l1);
37
38      return 0;
39  }
```

Listas V

```
40
41  void print_lst(list<int> l)
42  {
43      list<int>::iterator it;
44      for (it = l.begin(); it != l.end(); ++it)
45          cout << '\t' << *(it);
46      cout << '\n';
47  }
```

Listas VI

El método sort():

```
1  #include <iostream>
2  #include <iterator>
3  #include <list>
4  #include <cstdlib>
5  using namespace std;
6
7  void print_lst(list<int> );
8
9
10
11
12
13
14
```

Listas VII

```
15
16  int main()
17  {
18      list<int> l1;
19      for (int i = 0; i < 50; ++i)
20      {
21          l1.push_front(rand());
22      }
23      cout << "l1:"<<endl;
24      print_lst(l1);
25      l1.sort();
26      cout << "l1:"<<endl;
27      print_lst(l1);
28      return 0;
29  }
```

Listas VIII

```
30
31  void print_lst(list<int> l)
32  {
33      list<int>::iterator it;
34      for (it = l.begin(); it != l.end(); ++it)
35          cout <<*(it)<<'\n';
36      cout << '\n';
37  }
```

Stacks I

Las pilas son un tipo de contenedor adaptado, diseñado específicamente para operar en un contexto LIFO (último en entrar, primero en salir), donde los elementos se insertan y extraen solo de un extremo del contenedor.

Operaciones básicas con el contenedor stack :

- empty(): verifica cuando la fila está vacía
- size(): devuelve el tamaño de la stack
- push(): inserta un elemento
- pop(): borra un elemento
- top(): devuelve el primer elemento de la stack

Stacks II

```
1  #include <iostream>
2  #include <stack>
3  using namespace std;
4
5  int main()
6  {
7      stack<int> stack;
8      stack.push(1);
9      stack.push(2);
10     stack.push(4);
11     stack.push(5);
12     stack.pop();
13     stack.pop();
14
15
```

Stacks III

```
16      while (!stack.empty())
17      {
18          cout << stack.top() <<" ";
19          stack.pop();
20      }
21
22  return(0);
23  }
```

Queues I

Las queues son un tipo de contenedor adaptado, diseñado específicamente para operar como FIFOS (First Input First Output)

Las operaciones básicas con el contenedor queue son las mismas que con el de stack. La principal diferencia entre ambas es que el contenedor resuelve automáticamente en enlace entre nodos.

```
1 #include <iostream>
2 #include <queue>
3 using namespace std;
4
5 void print_queue(queue<int>);
6
7
8
```

Queues II

```
 9  int main()
10  {
11      queue<int> q;
12      q.push(10);
13      q.push(20);
14      q.push(30);
15      cout << "Impresion: ";
16      print_queue(q);
17      cout << "q.size()  : "  << q.size()  <<endl;
18      cout << "q.front() : " << q.front()<<endl;
19      cout << "q.back()  : "  << q.back()  <<endl;
20      q.pop();
21      print_queue(q);
22      return 0;
23  }
```

Queues III

```
24
25
26
27  void print_queue(queue<int> q)
28  {
29      queue<int> copy_queue=q;
30      while (!copy_queue.empty())
31      {
32          cout << '\t' << copy_queue.front();
33          copy_queue.pop();
34      }
35      cout << '\n';
36  }
```

Algoritmos I

La libería STL provee una multiplicidad de algoritmos para resolver problemas comunes y recurrentes en el ámbito de la ingeniería y el desarrollo de software.

```
1  #include <algorithm>
2  #include <iostream>
3  #include <vector>
4  using namespace std;
5
6  int main()
7  {    int ii;
8       vector<int> vect(10);
9       for(ii=0;ii<vect.size();ii++)
10           vect[ii]=rand();
```

Algoritmos II

```
11      cout << "Vector: "<<endl;
12      for(ii=0;ii<vect.size();ii++)
13          cout<< " "<<vect[ii]<<" "<<endl;
14
15      //Ordenamiento ascendente
16      sort(vect.begin(), vect.end());
17
18      cout << "Vector luego del ordenamiento: "<<endl;
19      for(ii=0;ii<vect.size();ii++)
20          cout << vect[ii] << " "<<endl;
21
22      //Ordenamiento descendente
23      sort(vect.begin(),vect.end(), greater<int>());
24
25      cout << "Vector luego del ordenamiento desc: "<<endl
```

Algoritmos III

```
26      for(ii=0;ii<vect.size();ii++)
27          cout << vect[ii]<< " "<<endl;
28
29      //Inversiin del arreglo
30      reverse(vect.begin(), vect.end());
31      cout << "\nVector invertido "<<endl;
32      for(ii=0;ii<vect.size();ii++)
33          cout << vect[ii]<< " "<<endl;
34
35      cout << "\nBusqueda de max y minimo "<<endl;
36      cout << *max_element(vect.begin(), vect.end())<<endl
37      cout << *min_element(vect.begin(), vect.end())<<endl
38
39      return 0;
40  }
```

¡Muchas gracias!

Consultas:

sperez@iua.edu.ar
drosso@iua.edu.ar