Clase 1 - C++ STL

Standard Template Library

Preparación GPC-UPC
Adaptado de las diapositivas de Rodolfo Mercado.
Complementado por Oscar Burga.
Tutor: Oscar Burga

Antes que nada...

- La clase de hoy probablemente sea pesada para muchos de ustedes
- No se asusten si tienen problemas para entender ahorita, sobretodo los chicos de Programación 1 y Programación 2. Con un poco de práctica todo empezará a cobrar sentido.
- El principal propósito de este PPT es que lo tengan de referencia para realizar este contest semanal que les estoy dejando *(está mucho más difícil que el de la semana anterior, y varios de los problemas pueden resolverse utilizando funciones y/o conceptos presentados en estas diapositivas)*.

Repaso: Tipos de dato

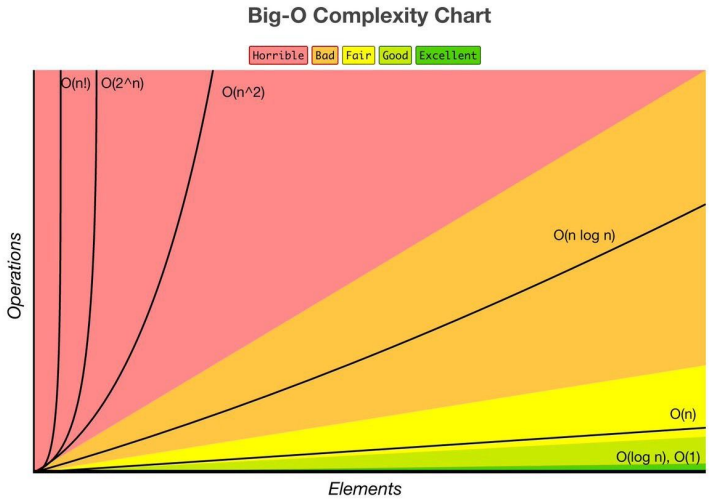
- Char: Caracter de 1 byte. Valores en el rango [0, 256)
- Short: Entero de 2 bytes. Límite aproximado de +- 2^15 (31000 aprox)
- Int: Entero de 4 bytes. Límite aproximado de +- 2x10^9
- Long Long: Entero de 8 bytes. Límite aproximado de +- 9x10^18
- Float: Decimal de 4 bytes. Precisión muy limitada. Eviten usarlo.
- Double: Decimal de 8 bytes. Precisión decente.
- Long double: Decimal de 8 bytes. Precisión decente.

Nota: Estos valores pueden variar de compilador a compilador, pero estos suelen ser los más estándares y los que se cumplen con el compilador de GCC/G++.

Overflow: Ocurre cuando nos excedemos del límite que puede soportar un tipo de dato. Se considera “comportamiento indefinido”. Puede causarles una respuesta equivocada!

Repaso: Análisis de algoritmos

- Notación Big-O
- Establece la cota superior para el crecimiento de una función
- Útil para medir y estimar el tiempo de ejecución de una solución/algoritmo
- También se puede utilizar para medir y estimar el consumo de memoria de nuestra solución
- Asumimos que las computadoras pueden realizar aproximadamente 10^8 operaciones por segundo.



Repaso: Análisis de algoritmos

```
int n, ans = 0;
cin >> n;
for(int i = 0; i<n; i++)        //Recorrido O(N)
    for(int j = i+1; j<n; j++)  //Recorrido O(N)
        ans+= f(i, j);          //Alguna función O(logN)
//Complejidad: O(N^2*logN)

for(int i = 0; i<pow(2, n); i++)//Recorrido O(2^N)
    ans = (ans+i)%100;          //Operaciones O(1)
//Complejidad: O(2^N)

//Complejidad total: O(2^N + N^2 logN)
//N^2*logN es insignificante a comparación de 2^N. Podemos obviar ese termino.
//Complejidad total: O(2^N)
```

Repaso: Análisis de algoritmos

```
for(int i = 0; i<pow(2, n); i++)//Recorrido O(2^N)
    for(int j = 0; j<n; j++)    //Recorrido O(N)
        ans = (ans + (i*j))%100;//Operaciones O(1)
//Complejidad: O(N*2^N)

//En este caso, no podemos obviar la N porque es una multiplicación
//y por lo tanto, es parte del termino de mayor grado.
//Complejidad total: O(N*2^N)
```

STL: Standard Template Library

- Librería de estructuras de datos y algoritmos que forman parte del estándar de C++.
- Evita que se tenga que programar algo de uso frecuente.
- Presenta conceptos como contenedores e iteradores.

```
vector<int> vec(100, -1);
```

```
pair<int,int> coords = make_pair(x, y);
```

```
sort(vec.begin(), vec.end());
```

```
queue<int> q;
```

```
stack<double> p;
```

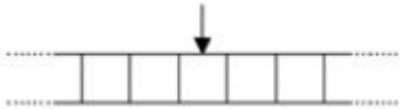
Contenedores

- Estructura que puede almacenar una colección de elementos del mismo tipo.
- Sus elementos se manejan casi siempre a través de iteradores (punteros).
- Son implementaciones de estructuras de datos muy comunes en programación.

Hoy empezaremos con los contenedores de secuencia (similares a los arreglos tradicionales, pero con algunas cualidades especiales), pero primero debemos ver qué es un iterador.

Iteradores

- Variable que apunta a un elemento de su contenedor. (Es un puntero)
- La gran mayoría de funciones y contenedores STL realizan sus operaciones mediante iteradores.
- De igual manera, muchas funciones STL retornan iteradores en vez de elementos.



Iteradores: Los Principales

Iterador al primer elemento del contenedor:

```
contenedor.begin();
```

Iterador a la primera posición fuera del contenedor:

```
contenedor.end();
```

Moverse por el contenedor:

```
iterator++, iterator--;
```

Acceder al valor apuntado por el iterador:

```
*(iterator);
```


```
vector<int> test = {1,2,3,4};
cout << *(test.begin()) << '\n'; //Imprime el primer elemento
cout << *(test.begin()+1) << '\n'; //Imprime el segundo elemento
```

Iteradores: Rangos

- Una subsegmento de un contenedor se denomina rango.
- Un rango se define como un intervalo cerrado en el inicio y abierto en el fin.

Bonus incluido: Dos funciones muy útiles de STL (en el futuro las veremos más detalladamente). Sort es O(NlogN). Reverse es O(N).

```
vector<int> test = {1,2,3,4};
sort(test.begin(), test.end());     //Ordena el vector dentro del rango [0, 5)
sort(test.begin(), test.begin()+3); //Ordena el vector dentro del rango [0, 3)
reverse(test.begin()+2, test.end());//Revierte el vector dentro del rango [2, 5)
```

Contenedores: Vector

- Contenedor que almacena elemento en posiciones contiguas de memoria.
- Pueden cambiar de tamaño en tiempo de ejecución.
- Permite insertar y eliminar un elemento al final en tiempo constante O(1).
- Permite acceso aleatorio (como un arreglo).

En términos prácticos, es un arreglo más *"cómodo"*.

Los **strings** (cadenas) son, esencialmente, vectores de caracteres, pero con algunas funciones adicionales muy útiles.

```
vector<int> test = {1,2,3,4};
cout << test[2] << endl; //3
```

```
string test = "abcde";
cout << test[2] << endl; //c
```

Vector y String: Funciones comunes principales

- **vec.size():** Retorna la cantidad de elementos en el contenedor. O(1)
- **vec.push_back(elem):** Agrega un elemento *elem* al final. O(1)
- **vec.pop_back()**: Elimina el elemento del final **(CUIDADO SI EL VECTOR ESTÁ VACÍO).** O(1)
- **vec.front():** Obtiene el primer elemento **(POR REFERENCIA)**. O(1)
- **vec.back():** Obtiene el último elemento **(POR REFERENCIA).** O(1)
- **vec.insert(it, elem):** Inserta un elemento *elem* en la posición apuntada por el iterador *it*. O(N)
- **vec.erase(it)**: Elimina el elemento en la posición apuntada por el iterador *it*. O(N)
- Varias de estas funciones tienen “sobrecargas” que les permiten funcionar de maneras distintas. Estas cosas se aprenderán solas con la práctica.

Por ahora, la que más usaremos será push_back() para facilitarnos la vida. Luego verán que pop_back() también puede ser extremadamente útil.

Problemitas de práctica

Es muy importante que vean y resuelvan estos problemas para familiarizarse con los vectores e iteradores, en especial si es su primera vez aprendiéndolos. Al inicio podrá parecer muy abstracto, confuso y difícil pero luego se darán cuenta de que en verdad es muy sencillo.

https://www.hackerrank.com/challenges/vector-sort/problem

https://www.hackerrank.com/challenges/vector-erase/problem

También pueden encontrarlos en el contest “Clase 1 - STL” en el Vjudge.
En el contest de la semana encontrarán también varios links donde pueden leer la documentación de las estructuras que vimos hoy.

Strings: Peculiaridades

Los strings tienen prácticamente todas las mismas funciones que vector y algunas adicionales. Las más importantes probablemente son:

- **S.find(T, pos)**: Busca el string *T* dentro del string *S* empezando desde la posición *pos*, y retorna la posición en la que inicia la ocurrencia de *T* en *S*. Si no se encuentra, retorna **string::npos**. O(S*T)
- **S.replace(pos, cnt, T):** Elimina los primeros *cnt* caracteres en la posición *pos* de la cadena *S*, e inserta la cadena T en su lugar. O(S + T)

```
string s = "test";
if (s.find("abc") == string::npos) cout << "No" << endl;

//string::npos casteado a entero con signo es -1
int pos = s.find("abc");
cout << pos << endl;

pos = s.find("es");
s.replace(pos, 2, "ab");
cout << s << '\n'; //"tabt";
```

Strings y caracteres: Trucos

- Funciones tolower(x) y toupper(x): Convierten el char *x* a minúscula o mayúscula, respectivamente (x debe representar una letra del alfabeto).
  Uso común: *x = tolower(x), x = toupper(x);*
- Los códigos ASCII son muy útiles. Pueden usarse como índices para arreglos, y muchas cosas más, y pueden realizar operaciones aritméticas con ellos (cuidado con el overflow).
  - Ejemplo: Contar ocurrencias de letras en una cadena

```
vector<int> cnt(256, 0);
string s = "abcdefa";

for(int i = 0; i<int(s.size()); i++)
    cnt[s[i]]++;

for(char i = 'a'; i<='z'; i++)
    cout << cnt[i] << ' ';
```

Pair: Muy útil, muy simple

- #include <utility>
- Como su nombre indica, es una estructura que guarda un par de elementos.
- Estos elementos pueden ser de cualquier tipo (incluso contenedores!)
- Metodos:
  - **p.first:** Acceso al primer elemento
  - **p.second:** Acceso al segundo elemento
  - **make_pair(a, b):** Crea un par con los elementos 'a' y 'b' como first y second, respectivamente.
- Aplicaciones comunes: Representar coordenadas, ordenar un arreglo guardando la posición original de cada elemento, etc.
- Ejemplos:

```
pair<int, int> coords;
pair<double, double> coords_decimales;
pair<int, vector<int>> Costo_y_Lista;
pair<pair<int,int>, pair<int,int>> Par_de_pares_de_enteros;
```