

## Parte I – Sintaxe da Linguagem

### Estrutura do programa C++

```
/*  
    bloco de comentários  
*/  
  
// comentário de linha  
  
#include <iostream>  
#include <iomanip>  
  
using namespace std;  
  
tipo_de_retorno nome_da_função(lista_de_parâmetros)  
{  
    corpo da função  
    return valor; // ou return; se função for void  
}  
  
int main()  
{  
    return 0;  
}
```

### Diretiva #include

Inclui arquivo da biblioteca padrão

```
#include <nome do arquivo>
```

Inclui outros arquivos

```
#include "nome do arquivo"
```

### Tipos de dados

Os seguintes são os tipos mais comuns para definição de variáveis e campos de estruturas. Os tipos inteiro (int) por padrão são sinalizados (*signed*), ou seja, podem receber valores positivos e negativos. Para definir uma variável sem sinal adicione a palavra reservado *unsigned*.

Tipo	Descrição	Exemplo
char	Um único caracter	char opcao='S';
int	Número inteiro de tamanho padrão	int i = -1;
short int	Número inteiro curto	unsigned short int n=10U;
long int	Número inteiro longo	long int i=500000;
long long int	Número inteiro mais longo	long long int i=100000000LL;
float	Número em ponto flutuante com precisão simples	float var=15.54;
double	Número em ponto flutuante com precisão dupla	double pi=3.14159265;
bool	Valor lógico (booleano)	bool flag=false;
string*	Sequencia de caracteres (texto)	string s="aeiou";

\*: Não é um tipo fundamental, é uma *classe*. Depende de #include <string> ou outro que indiretamente inclua este arquivo

## Inferência de tipo

Usando a palavra reservada *auto* na definição de uma variável que está sendo inicializada, o compilador irá inferir automaticamente o seu tipo.

```
auto stopped=false; // stopped é definido automaticamente como um bool
```

## Inicialização de variáveis

Inicialização é a especificação de um valor inicial para uma variável, no momento em que ela está sendo definida. Existem diversas sintaxes que podem ser usadas para isto.

Sintaxe	Exemplo
<pre>tipo nome=valor; // Estilo C tipo nome(valor); // Estilo C++ antigo tipo nome{valor} // Estilo C++11. Não usar com <i>auto</i></pre>	<pre>int n=10; char letra('A'); string s{"aeiou"}; bool f{false};</pre>

## Constantes

Uma constante é um dado que não pode ter seu valor modificado em tempo de execução. A definição é similar à de uma variável, exceto pela palavra reservada *const* e que a inicialização é obrigatória. Ao contrário de variáveis, constantes são definidas com frequência no escopo global.

```
const float NOTA_MAXIMA{10.0}; // É comum definir constantes em letras maiúsculas
```

## Arrays

Conjunto de elementos do mesmo tipo. Podem ser unidimensional (vetor) ou multidimensional (matrizes). Sempre o primeiro índice é [0] (zero) e o último é [N-1] (N é o tamanho do array)

Sintaxe	Exemplo
<pre>tipo nome[tamanho]; tipo nome[linhas][colunas];</pre>	<pre>int vet[10]; char matriz[5][100];</pre>

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

const int N{5};

float media(int v[]) {
    float soma{0.0};

    for (int i=0; i<N; i++) {
        soma += v[i];
    }
    return soma/N;
}

void leitura(int v[]) {
    for (int i=0; i<N; i++) {
        cout << "[" << i << "]: ";
        cin >> v[i];
    }
}

int main() {
    int vet[N];
    leitura(vet);
    cout << "Media: " << media(vet) << endl;
    return 0;
}
```

Arrays podem ser inicializados com uma lista de valores separados por vírgula:

```
char vogais[5]{'a', 'e', 'i', 'o', 'u'};
```

Strings podem ser acessadas como um array de caracteres

```
string s{"aeiou"};
s[0] = 'A';
cout << s[1]; // Exibe o segundo caracter
```

## Entrada (cin) e Saída (cout)

#include <iostream>

Sintaxe cin >> variavel; cout << valor;  Para strings (com brancos): getline(cin, variável_string);  Usar quando misturar >> com getline() para tirar o ENTER do teclado: cin.ignore();	Exemplo string nome; int n; cout << "N:"; cin >> n; cout << n << endl;  cout << "Nome:"; cin.ignore(); getline(cin, nome); cout << nome;
-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------	------------------------------------------------------------------------------------------------------------------------------------------------------------------------

## Manipuladores de saída

#include <iomanip>

Usados para formatação da saída de dados.

fixed: quantidade fixa de casas decimais setprecision(n): número de casas decimais; setw(n): tamanho reservado para exibir o valor; setfill(c); caracter para preencher o espaço reservado não ocupado pelo valor;	Exemplo float f{1.5}; int n{33}; cout << fixed << setprecision(2) << f; cout << endl; cout << setw(5) << setfill('0') << n;
-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------	--------------------------------------------------------------------------------------------------------------------------------------------

## Acertando a acentuação no Code::Blocks

#include <locale>

```
#include <iostream>
#include <locale>

using namespace std;

int main()
{
    // Define localização do programa para português
    setlocale(LC_ALL, "Portuguese");

    cout << "Olá Mundão!" << endl;

    return 0;
}
```

## Operadores aritméticos

Operador	Descrição	Exemplo
=	Atribuição	a = b = c = d =0;
+	Soma	a = b + 1;
-	Subtração	x = y- z;
*	Multiplicação	cubo = n * n;
/	Divisão (com valores int, o resultado não terá casas decimais)	a = b / 0.5;
%	Resto da divisão inteira	if(num%2==0) par=true;
++	Incremento (soma 1)	I++; ou ++i;
--	Decremento (subtrai 1)	n--; ou --n;

## Operadores combinados

Expressão Normal	Exemplo Simplificado
a = a + b;	a+=b;
a = a - b;	a-=b;
a = a * b;	a*=b;
a = a / b;	a/=b;
a = a % b;	a%=b;

## Operadores relacionais

Operador	Descrição
>	Maior que
>=	Maior ou igual à
<	Menor que
<=	Menor ou igual à
==	Igual à
!=	Diferente de

## Operadores lógicos

Operadores	Descrição	Exemplo
&& and	AND lógico	if(a>b && b!=0)
or	OR lógico	if(a==b or a==c)
! not	NOT lógico	if(!flag)

## Moldagem ou cast

Converte valor de um tipo para outro. Não usar para conversão entre classes. Para converter string para número ou vice-versa ver as funções apropriadas mais adiante.

Sintaxe	Exemplo
<code>(tipo) valor // Estilo C</code> <code>tipo(valor) // Estilo funcional</code>	<code>media = (float)soma/n;</code> <code>media = float(soma)/n;</code>

## Operador ternário

Avalia *condição*, se verdadeira a expressão assume *valor1* senão *valor2*.

Sintaxe	Exemplo
<code>(condição)?valor1:valor2</code>	<code>maior = (a&gt;b) ? a : b;</code>

## Comando if

Executa comandos condicionalmente. O *else* é opcional.

Sintaxe	Exemplo
<code>if(condição){</code> Executa se condição é verdadeira <code>}</code> <code>else{</code> Executa se condição é falsa <code>}</code>	<code>if(n&gt;0)</code> <code>vet[i] = n;</code> <code>else{</code> <code>cout &lt;&lt; "Inválido!";</code> <code>return 1;</code> <code>}</code>

## Comando do while

Repete enquanto condição for verdadeira, testando no final do laço.

Sintaxe	Exemplo
<code>do{</code> comandos a serem repetidos <code>}while(condição);</code>	<code>do{</code> <code>cout &lt;&lt; "N:";</code> <code>cin &gt;&gt; n;</code> <code>}while(n&lt;0);</code>

## Comando while

Repete enquanto condição for verdadeira, testando no início do laço.

Sintaxe	Exemplo
<code>while(condição){</code> comandos a serem repetidos <code>}</code>	<code>int i{10};</code> <code>while(i&gt;0){</code> <code>cout &lt;&lt; i &lt;&lt; endl;</code> <code>i--;</code> <code>}</code>

## Comando continue e break

O comando `continue` pode ser utilizado dentro de laços para ignorar o restante da passagem atual. Os comandos abaixo do `continue`, não serão executados e o programa passa para o teste da condição do laço atual. Já o comando `break`, quando utilizado em um laço, finaliza o laço atual, pulando para o primeiro comando após o final dele.

Sintaxe	Exemplo
<pre>break;  continue;</pre>	<pre>while(true){     cin &gt;&gt; n;     if(n==0) break;     if(n&lt;0) continue;     calculo(n); }</pre>

## Comando for

Cria um laço que contém um conjunto de comandos que será executado um número fixo de vezes.

Sintaxe	Exemplo
<pre>for(inicialização; condição; atualização) {     comandos a serem repetidos }</pre>	<pre>int vet[10]; for(int i=0; i&lt;10; i++){     cin &gt;&gt; vet[i]; }</pre>

A partir do C++11 existe uma sintaxe mais curta que pode ser usada para iterar um intervalo de valores. É o *range-for*.

Sintaxe	Exemplo
<pre>for(auto var : intervalo){     comandos a serem repetidos }</pre>	<pre>int vet[5]{10, 20, 30, 40, 50}; for(auto n : vet){     cout &lt;&lt; n; }</pre>

## Comando switch

Compara uma variável ou expressão com diversos valores diferentes, fornecidos em cada um dos `case`. Se for encontrado um valor igual, todos os comandos após este `case` são executados, até encontrar um `break`. O `default`, no final do comando, é opcional. Os comandos após o `default` somente serão executados se o valor não coincidir com nenhum dos valores correspondentes aos `case` anteriores. Atenção: Somente pode ser usado com tipos inteiros.

Sintaxe	Exemplo
<pre>switch(var){     case valor1:         comandos a executar se var==valor1         break;     case valor2:         comandos a executar se var==valor2         break;     default:         caso nenhum dos valores }</pre>	<pre>int dia; cin &gt;&gt; dia; switch(dia){     case 1: cout &lt;&lt; "Domingo\n";         break;     case 2: cout &lt;&lt; "Segunda-feira\n";         break;     case 3: cout &lt;&lt; "Terça-feira\n";         break;     case 4: cout &lt;&lt; "Quarta-feira\n";         break;     case 5: cout &lt;&lt; "Quinta-feira\n";         break;     case 6: cout &lt;&lt; "Sexta-feira\n";         break;     case 7: cout &lt;&lt; "Sábado\n";         break;     default:         cout &lt;&lt; "Dia inválido\n"; }</pre>

## Passagem por valor X por referência

- **Passagem de Parâmetro por Valor:** Uma cópia do valor passado pela função chamadora é fornecido para a função chamada. Modificações no parâmetro recebido feita na função chamada não afetarão o valor da variável fornecida na função chamadora.
- **Passagem de Parâmetro por Referência:** Em vez de passar uma cópia do valor de uma variável como argumento de uma função, pode ser passado uma referência a ela. neste caso, qualquer alteração feita usando a referência irá modificar o valor da variável utilizada na chamada da função. Um **&** antes do nome do parâmetro indica que será usado passagem por referência. A ausência deste símbolo denota passagem por valor.

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

// Primeiro parâmetro por referência, segundo por valor
void dobro(int &r, int v){
    r *=2;
    cout << "Dobro: " << r << endl; // Mostra 10
    v *=2;
    cout << "Dobro: " << v << endl; // Mostra 12
}

int main(){
    int a{5}, b{6};
    dobro(a, b); // O valor de A é modificado na função. B não é.
    cout << "A: " << a << " B: " << b << endl; //Mostra 10 e 6
    return 0;
}
```

## Estruturas

Uma estrutura é o conjunto de variáveis agrupadas sob um nome único, sendo que estas variáveis podem ser de tipos de dados diferentes. A estrutura serve para organizar, de forma lógica, algum dado cujo valor é composto por mais de uma variável.

O uso de structs em um programa C++ passa por estas etapas:

- Definição da estrutura, normalmente antes de qualquer função do programa, onde define-se o seu nome e os tipos e nomes de cada um dos seus campos.
- Definição das variáveis: dentro das funções que precisarem fazer uso de variáveis cujo tipo corresponde à estrutura definida. Esta variável pode ser inicializada, na definição, colocando-se os valores iniciais de cada um dos seus campos, na ordem, entre um par de chaves e separados por vírgula.
- O uso das variáveis: para acessar o valor dos campos de uma variável de estrutura separa-se o nome da variável do nome do campo com um ponto (`variavel.campo` ou `vetor[i].campo`).

Sintaxe	Exemplo
<pre>struct nome {     tipo membro_1;     tipo membro_2;     tipo membro_n; };</pre>	<pre>// definindo a struct struct data{     int dia;     int mes;     int ano; };</pre>

```

#include <iostream>
#include <iomanip>
using namespace std;

struct aluno {
    int matricula;
    string nome;
    float nota;
};

void exibealuno(aluno a) {
    cout << setfill('0') << setw(5) << a.matricula << " ";
    cout << left << setfill(' ') << setw(30) << a.nome << " ";
    cout << right << setw(5) << fixed << setprecision(2) << a.nota << endl;
}

int main()
{
    aluno turma[2]{ {999123, "Fulano de Tal", 9.5},
                    {999063, "Cicrano", 5.0}
    };

    for (auto a : turma)
        exibealuno(a);
    return 0;
}

```

## Ponteiros

Um ponteiro é uma variável definida para armazenar o endereço de memória onde está armazenada outra variável. Um asterisco (\*) antes do nome de uma variável define-a como sendo um ponteiro. A constante `nullptr` pode ser empregada para indicar que um ponteiro não aponta para nada. Se usado em alguma condição um ponteiro igual a `nullptr` é considerado um falso.

Sintaxe tipo *nome	Exemplo int *pti; data *pd; char *s{nullptr};
-----------------------	--------------------------------------------------------

## Alguns operadores para ponteiros

Sintaxe	Descrição	Exemplo
&var	Fornecer o endereço de memória onde está armazenada uma variável.	int a{5}; int *p; p = &a; cout << *p; *p = 10;
*ptr	Valor armazenado na variável apontada por um ponteiro.	
ptr->campo	Acessa um campo da estrutura apontada pelo ponteiro.	data dt, *p{&dt}; p->dia = 10;

Por alocação dinâmica de memória entende-se a tarefa de reservar memória do computador para o armazenamento de dados durante a execução do programa (em tempo de execução). Com esta técnica o programa, e portanto o programador, tem a possibilidade de decidir o momento e a quantidade de memória a ser reservada para programa, além daqueles dados automaticamente alocados pelo compilador. Para usar esta técnica, deve-se definir um ponteiro que receberá o endereço da memória alocada. O acesso a esta memória deve ser feito por intermédio de ponteiros. Quando estes dados não forem mais necessários, deve-se liberar a memória para o sistema.

Sintaxe	Descrição
new	Aloca um valor e retorna o endereço da memória reservada.
delete	Libera a memória ocupado por um valor dinamicamente alocado com new.
new T[]	Aloca um array (vetor) do tipo T e retorna o endereço da memória reservada.
delete[]	Libera a memória reservada para um array dinamicamente alocado.
(nothrow)	Usado com o operador new, possibilita verificar se a alocação teve sucesso ou falhou. Caso não seja possível reservar a memória solicitada o ponteiro receberá nullptr.

```
#include <new>
#include <iostream>

using namespace std;

struct data {
    int dia, mes, ano;
};

int main()
{
    // Aloca dinamicamente uma data
    data *natal = new data;
    natal->dia=25; natal->mes=12; natal->ano=2015;
    cout << natal->dia << '/' << natal->mes << '/' << natal->ano << endl;
    // Deleta a variavel apontada da memória (não deleta o ponteiro)
    delete natal;

    int n;
    cout << "Tamanho do vetor: "; cin >> n;
    int *v = new (nothrow) int[n]; // Aloca dinamicamente um vetor
    if(v==nullptr) { // Poderia ser: if(!v){
        cout << "Falhou\n";
        return 1;
    }
    for(int i=0; i<n; i++)
        v[i] = 1; // Atribui 1 para todo o vetor

    delete[] v; // Deleta todo o vetor da memoria
    return 0;
}
```



## Parte II – Resumo da Biblioteca Padrão

### Funções Matemáticas

Funções	Descrição	Include
abs(x)	Calcula o módulo ou valor absoluto de x (x sem sinal)	<cstdlib> para inteiro <cmath> para ponto-flutuante
sin(x)	Calcula o seno de x	<cmath>
cos(x)	Calcula o cosseno de x	<cmath>
tan(x)	Calcula a tangente de x	<cmath>
log(x)	Calcula o logaritmo natural de x	<cmath>
pow(x, y)	Calcula x elevado à y-ésima potência: $x^y$	<cmath>
sqrt(x)	Calcula a raiz quadrada de x	<cmath>

```
#include <iostream>
#include <cmath>

using namespace std;

int main ()
{
    double x;
    cout << "Informe X: ";
    cin >> x;
    cout << "abs(x)   = " << abs (x) << endl;
    cout << "sin(x)    = " << sin (x) << endl;
    cout << "cos(x)    = " << cos (x) << endl;
    cout << "tan(x)    = " << tan (x) << endl;
    cout << "log(x)    = " << log (x) << endl;
    cout << "sqrt(x)   = " << sqrt (x) << endl;
    cout << "Cubo de x  = " << pow (x, 3) << endl;
    return 0;
}
```

### Funções de teste e conversão de caracter

#include <cctype>

Funções	Descrição
isalnum(c)	Retorna verdadeiro se o caracter passado por parâmetro é uma letra ou um dígito numérico.
isalpha(c)	Retorna verdadeiro se o caracter passado por parâmetro é uma letra.
isdigit(c)	Retorna verdadeiro se o caracter passado por parâmetro é um dígito numérico.
islower(c)	Retorna verdadeiro se o caracter passado por parâmetro é uma letra minúscula.
isupper(c)	Retorna verdadeiro se o caracter passado por parâmetro é uma letra maiúscula.
isspace(c)	Retorna verdadeiro se o caracter passado por parâmetro é um espaço em branco. O caracter correspondente à barra de espaços, quebra de linha e tabulações são considerados brancos.
tolower(c)	Se o caracter for uma letra, retorna-o convertido para minúsculo, senão retorna-o sem alteração.
toupper(c)	Se o caracter for uma letra, retorna-o convertido para maiúsculo, senão retorna-o sem alteração.

```

#include <iostream>
#include <cctype>
using namespace std;

int main() {
    string s;
    cout << "String: "; getline(cin, s);
    cout << "Exibindo somente as letras, em maiusculo:\n";
    for (char c : s)
        if (isalpha(c)){
            c = toupper(c);
            cout << c;
        }
    return 0;
}

```

## E/S em arquivos (*streams*)

**#include <fstream>**

Todo arquivo precisa ser aberto para que o seu conteúdo esteja disponível ao programa. A ação de abrir o arquivo envolve a solicitação do acesso ao sistema operacional e reservar áreas de memória para armazenamento temporário de dados necessários à transferência. Após a abertura, se esta teve sucesso, o programa pode ler ou escrever dados no arquivo aberto, através do *stream* (fluxo de E/S) obtido. Eventualmente a abertura pode falhar, como nos casos em que o arquivo a ser lido não existe, o usuário não tem permissão de acesso ao arquivo ou diretório, entre outros. Finalmente, quando o programa não necessitar mais acessar o conteúdo do arquivo, este deve ser fechado.

Sintaxe	Descrição
<code>ifstream arq;</code>	Define um arquivo de entrada
<code>arq.open(nome)</code>	Solicita a abertura do arquivo cujo nome foi passado por parâmetro
<code>if(!arq)</code>	Testa se a abertura do arquivo ou última operação leitura/escrita falhou.
<code>arq &gt;&gt; var;</code>	Lê um valor do arquivo-texto e armazena-o na variável
<code>getline(arq, str);</code>	Lê uma linha de um arquivo-texto e armazena-a na string
<code>arq.get(c);</code> ou <code>c=arq.get();</code>	Lê um caracter de um arquivo-texto e armazena-o na variável char passada por parâmetro ou Lê um caracter de um arquivo-texto e retorna o caracter lido
<code>arq.close();</code>	Fecha o arquivo aberto anteriormente
<code>ofstream arq;</code>	Define um arquivo de entrada
<code>arq &lt;&lt; var;</code>	Escreve o valor da variável no arquivo

```
// Exibe o conteúdo de um arquivo-texto, um caractere por vez
#include <iostream>
#include <fstream>

using namespace std;

int main() {
    ifstream arq;
    arq.open("teste.txt");
    if (!arq) {
        cout << "Nao abriu\n";
        return 1;
    }

    char c;
    while(true){
        c = arq.get();

        if (!arq) break;
        cout << c;
    }
    arq.close();
    return 0;
}
```

```
// Exibe o conteúdo de um arquivo-texto, uma linha por vez
#include <iostream>
#include <fstream>

using namespace std;

int main() {
    ifstream arq;
    string s;
    cout << "Nome do arquivo: "; getline(cin, s);

    arq.open(s);
    if (!arq) {
        cout << "Nao abriu\n";
        return 1;
    }

    while(getline(arq, s)){
        cout << s << endl;
    }
    arq.close();
    return 0;
}
```

```
// Lê numeros inteiros de um arquivo-texto e grava n*2 em outro arquivo
#include <iostream>
#include <fstream>

using namespace std;

int main()
{
    string s;
    ifstream arq;
    cout << "Nome do arquivo de entrada: "; getline(cin, s);
    arq.open(s);
    if (!arq) {
        cout << "Nao abriu " << s << endl;;
        return 1;
    }

    ofstream saida;
    cout << "Nome do arquivo de saida: "; getline(cin, s);
    saida.open(s);
    if (!saida) {
        cout << "Nao abriu " << s << endl;;
        return 1;
    }
    int n;
    while(arq >> n) {
        saida << n*2 << endl;
    }
    arq.close();
    saida.close();
    return 0;
}
```

## Parte III – Resumo da *Standard Template Library* (STL)

### O Container vector

**#include <vector>**

Trata-se de um array dinâmico, isto é, um vetor cujo tamanho se ajusta automaticamente conforme novos elementos são inseridos, sendo conveniente apenas para inserção no final.

Um vector pode ser acessado aleatoriamente (`v[p]`) ou sequencialmente. Neste segundo caso recomenda-se o uso de iteradores (`iterator`) ou *range-for*.

Definições básicas	Exemplo
<pre>// Substituir T pelo tipo que se quer armazenar vector&lt;T&gt; nome;      // Inicialmente vazio vector&lt;T&gt; nome(n);   // Criado com n elementos</pre>	<pre>vector&lt;float&gt; v;</pre>
<pre>// Iterator para percorrer sequencialmente vector&lt;T&gt;::iterator it;</pre>	<pre>vector&lt;float&gt;::iterator it;</pre>
<pre>// Iterator para percorrer em ordem reversa vector&lt;T&gt;::reverse_iterator rit;</pre>	<pre>vector&lt;float&gt;::reverse_iterator rit;</pre>

Principais funções		Descrição
<code>v.push_back(valor)</code>		Insere valor no final do vector v
<code>v.pop_back()</code>		Deleta o último elemento em v
<code>v.size()</code>		Retorna o tamanho (quantidade de elementos) do vetor v
<code>v.empty()</code>		Retorna um bool indicando se v está vazio (nenhum elemento)
<code>v.clear()</code>		Deleta todo o conteúdo do vetor v
<code>v.begin()</code>		Retorna um iterator que aponta para o início do vetor v
<code>v.end()</code>		Retorna um iterator que aponta após o final do vetor v
<code>v.rbegin()</code>		Retorna um iterator reverso que aponta para o último elemento em v
<code>v.rend()</code>		Retorna um iterator reverso que aponta para antes do início de v

```

#include <iostream>
#include <vector>

using namespace std;

int main()
{
    vector<int> vet;                // Cria um vector vazio de inteiros

    vet.push_back(10);             // Insere 3 valores no final
    vet.push_back(20);
    vet.push_back(30);

    cout << "Percorre com []\n";
    for(int i=0; i<vet.size(); i++)
        cout << vet[i] << endl;

    cout << "Percorre com range-for\n";
    for(auto v : vet)
        cout << v << endl;

    cout << "Percorre com ::iterator\n";
    for(auto it=vet.begin(); it<vet.end(); ++it)
        cout << *it << endl;

    vet.pop_back();               // Retira o último valor
    cout << "Percorre com ::reverse_iterator\n";
    for(auto ir=vet.rbegin(); ir<vet.rend(); ++ir)
        cout << *ir << endl;
    return 0;
}

```

## O Container list

#include <list>

Container genérico correspondente à uma lista duplamente encadeada. Como listas encadeadas não permitem acesso aleatório, elas podem ser percorrida apenas sequencialmente com iterators ou *range-for*.

### Definições básicas

```

// Substituir T pelo tipo que se quer armazenar
list<T> nome;

```

### Exemplo

```

struct data{
    int dia, mes, ano;
};

list<data> ls;

```

Principais funções	Descrição
ls.push_front(valor)	Insere valor no início da lista ls.
ls.push_back(valor)	Insere valor no final da lista ls.
ls.insert(pos, valor)	Insere valor na lista antes do nodo apontado pelo iterator pos.
ls.pop_front()	Deleta o valor que está no início da lista. Não retorna o valor retirado.
ls.pop_back()	Deleta o valor que está no final da lista. Não retorna o valor retirado.
ls.remove(valor)	Deleta da lista o(s) nodo(s) que contêm o valor passado por parâmetro.
ls.remove_if(fn)	
ls.erase(it)	Deleta da lista ls o nodo apontado pelo iterator it.
ls.erase(inicio, fim)	Deleta todos os nodos situados no intervalo entre os iterator inicio e fim
ls.clear()	Deleta todos os nodos da lista.
ls.front()	Retorna uma referência para o valor que está no início da lista.
ls.back()	Retorna uma referência para o valor que está no final da lista.
ls.size()	Retorna o tamanho (quantidade de elementos) contidos na lista.

ls.empty()	Retorna um bool indicando se ls está vazia (nenhum elemento).
ls.sort()	Ordena os valores contidos em ordem crescente.
ls.begin()	Retorna um iterator apontando para o primeiro elemento da lista.
ls.end()	Retorna um iterator que aponta após o final da lista.
ls.rbegin()	Retorna um iterator reverso que aponta para o último elemento em ls.
ls.rend()	Retorna um iterator reverso que aponta para antes do início de ls.

```
#include <iostream>
#include <list>

using namespace std;

// Passar sempre containers por referência para não fazer cópia de todos os
// dados
// Se a função não for modificar os dados, passar por referência constante
void mostra(const list<string>& l)
{
    for(auto s: l) cout << s << endl;
}

int main()
{
    list<string> l;                // Lista encadeada de strings

    cout << "Inserindo nas extremidades\n";
    l.push_back("a");             // Insere no fim
    l.push_front("b");            // Insere no inicio
    l.push_back("c");             // Insere no fim
    l.push_front("d");            // Insere no inicio
    mostra(l);

    cout << "Retirada das extremidades\n";
    l.pop_front(); // Retira do inicio
    l.pop_back();  // Retira do fim
    mostra(l);

    cout << "Inserindo antes do segundo valor\n";
    auto it=l.begin();           // Aponta para o primeiro
    ++it;                        // Avança para o próximo
    l.insert(it, "e");            // Insere antes do segundo
    mostra(l);

    cout << "Retira um valor\n";
    int tam=l.size();             // Guarda o tamanho da lista
    l.remove("x");                // Retira o valor
    if(tam!=l.size()) cout << "Retirou\n"; // Se mudou o tamanho
    else cout << "Nao retirou\n";
    mostra(l);

    cout << "Ordena a lista\n";
    l.sort();
    mostra(l);
    return 0;
}
```

## O Container stack

**#include <stack>**

Implementa uma pilha que armazena nodos contendo um tipo genérico qualquer. Por ser uma estrutura LIFO não pode ser acessada com iterators.

Definições básicas	Exemplo
<code>stack&lt;T&gt; nome;</code>	<code>// Substituir T pelo tipo</code>
	<code>stack&lt;int&gt; p;</code>

Principais funções	Descrição
<code>p.push(valor)</code>	Insere valor no topo da pilha <i>p</i>
<code>p.pop()</code>	Deleta o valor que está no topo da pilha <i>p</i> . Não retorna o valor retirado.
<code>p.top()</code>	Retorna o valor que está no topo da pilha. Corresponde à operação de consulta na pilha.
<code>p.size()</code>	Retorna o tamanho (quantidade de elementos) contidos em <i>p</i> .
<code>p.empty()</code>	Retorna um bool indicando se <i>p</i> está vazia (nenhum elemento)

```
#include <iostream>
#include <stack>

using namespace std;

int main()
{
    stack<string> pilha;           // Define uma pilha de strings

    pilha.push("passo fundo");    // Empilha 3 valores
    pilha.push("carazinho");
    pilha.push("sarandi");

    while(!pilha.empty()){        // Enquanto a pilha não estiver vazia
        cout << pilha.top() << endl; // Exibe o valor do topo
        pilha.pop();              // Retira um valor
    }
    return 0;
}
```

## O Container queue

**#include <queue>**

Container genérico correspondente à uma fila. Por ser uma estrutura FIFO não pode ser acessada com iterators.

Definições básicas	Exemplo
<code>// Substituir T pelo tipo que se quer armazenar</code>	
<code>queue&lt;T&gt; nome;</code>	<code>queue&lt;int&gt; f;</code>

Principais funções	Descrição
<code>f.push(valor)</code>	Insere <i>valor</i> no final da fila <i>f</i>
<code>f.pop()</code>	Deleta o valor que está na frente da fila <i>f</i> . Não retorna o valor retirado.
<code>f.front()</code>	Retorna o valor que está na frente da fila. Corresponde à operação de consulta na fila.
<code>f.back()</code>	Consulta o valor que está na ré da fila.
<code>f.size()</code>	Retorna o tamanho (quantidade de elementos) contidos na fila <i>f</i> .
<code>f.empty()</code>	Retorna um bool indicando se <i>f</i> está vazia (nenhum elemento)



```

#include <iostream>
#include <queue>

using namespace std;

int main()
{
    queue<float> fila;          // Define uma fila de inteiros

    fila.push(1.5);            // Insere 4 valores na fila
    fila.push(3.4);
    fila.push(0.8);
    fila.push(9.2);

    cout << "Frente: " << fila.front() << endl;    // Consulta a frente
    cout << "Re: " << fila.back() << endl << endl; // Consulta a ré

    while(fila.size() > 0){          // Enquanto o tamanho for maior que 0
        cout << "Frente: " << fila.front() << endl; // Exibe frente
        fila.pop();                  // Retira um valor da fila
    }
    return 0;
}

```

## Algoritmos

#include <algorithm>

Principais funções	Descrição
find(inicio, fim, val)	Pesquisa sequencial: Retorna um iterator para a primeira ocorrência de <i>val</i> dentro do intervalo entre <i>inicio</i> (inclusive) e <i>fim</i> (exclusive). Se não encontrar, retorna o iterator <i>fim</i> .
binary_search(inicio, fim, val)	Pesquisa binária: Retorna um bool indicando se <i>val</i> existe ou não dentro do intervalo <b>ordenado</b> [ <i>inicio</i> , <i>fim</i> ).
sort(inicio, fim)	Ordena em ordem crescente os valores contidos dentro do intervalo [ <i>inicio</i> , <i>fim</i> ).
sort(inicio, fim, func)	Ordena em ordem crescente os valores contidos dentro do intervalo [ <i>inicio</i> , <i>fim</i> ] usando a função de comparação fornecida.
count(inicio, fim, val)	Retorna a quantidade de valores contidos no intervalo [ <i>inicio</i> , <i>fim</i> ] que são iguais a <i>val</i> .
copy(inicio, fim, destino)	Copia todos os valores no intervalo [ <i>inicio</i> , <i>fim</i> ] para <i>destino</i>
copy_n(inicio, n, destino)	Copia todos <i>n</i> valores a partir de <i>inicio</i> para <i>destino</i>
fill(inicio, fim, val)	Atribui <i>val</i> para todos os valores no intervalo [ <i>inicio</i> , <i>fim</i> )
replace(inicio, fim, a, b)	Substitui toda a ocorrência de <i>a</i> por <i>b</i> dentro do intervalo [ <i>inicio</i> , <i>fim</i> )
max(a, b)	Retorna o maior valor de <i>a</i> e <i>b</i> , ou <i>a</i> se ambos são iguais.
max_element(inicio, fim)	Retorna um iterator para o maior valor contido no intervalo [ <i>inicio</i> , <i>fim</i> )
min(a, b)	Retorna o menor valor de <i>a</i> e <i>b</i> , ou <i>a</i> se ambos são iguais.
min_element(inicio, fim)	Retorna um iterator para o menor valor contido no intervalo [ <i>inicio</i> , <i>fim</i> )

## Algoritmos Numéricos

#include <numeric>

Principais funções	Descrição
accumulate(inicio, fim, val)	Retorna a soma dos valores no intervalo [ <i>inicio</i> , <i>fim</i> ) partindo do valor inicial <i>val</i>
iota(inicio, fim, val)	Atribui valores sequenciais para o intervalo [ <i>inicio</i> , <i>fim</i> ) partindo do valor inicial <i>val</i>

Principais funções	Descrição
swap(a, b)	Troca o valor de a por b.

```
#include <algorithm>
#include <iostream>
#include <iterator> // Para as funções begin() e end()

using namespace std;

int main()
{
    int va[] {98, 50, 37, 0, -1, 50, 33, 100};

    sort(begin(va), end(va)); // Ordena o array
    cout << "Valores ordenados\n";
    for(auto v : va)
        cout << v << ' ';

    cout << "\nPesquisar: ";
    int val;
    cin >> val;

    if(binary_search(begin(va), end(va), val)) // Pesquisa binária
        cout << "Achou\n";
    else
        cout << "Nao achou\n";

    auto it = find(begin(va), end(va), val); // Pesquisa sequencial
    if(it!=end(va)) {
        *it += 1000;
        cout << "Somou 1000 na primeira ocorrencia\n";
        for(auto v : va)
            cout << v << ' ';
    }

    it = max_element(begin(va), end(va));
    cout << "\nMaior valor: " << *it << endl;

    int vb[10];
    // Copia os 4 primeiros valores de va para vb
    copy_n(begin(va), 4, begin(vb));
    fill(begin(vb)+4, end(vb), 0); // Atribui 0 a partir de vb[4]
    cout << "Segundo vetor: \n";
    for(auto v : vb)
        cout << v << ' ';

    int n=count(begin(vb), end(vb), 0); // conta os valores==0
    cout << "\nEncontrou " << n << " zero(s)\n";

    val = accumulate(begin(vb), end(vb), 0);
    cout << "Soma de vb: " << val << endl;

    iota(begin(vb), end(vb), 500); // vb[0]=500; vb[1]=501; ...
    for(auto v : vb)
        cout << v << ' ';
    return 0;
}
```

```

#include <iostream>
#include <iomanip>
#include <vector>
#include <algorithm>

using namespace std;

struct data {
    int dia, mes, ano;
};

void exibedata(data dt)
{
    cout << setw(2) << setfill('0') << dt.dia << "/" <<
        setw(2) << setfill('0') << dt.mes << "/" <<
        setw(4) << setfill('0') << dt.ano << endl;
}

void lerdata(data &dt)
{
    cin >> dt.dia;    cin.ignore(); // Descarta o '/'
    cin >> dt.mes;    cin.ignore(); // Descarta o '/'
    cin >> dt.ano;
}

bool ordena(data a, data b)
{
    if(a.dia < b.dia) return true;
    return false;
}

int main()
{
    vector<data> dts;
    data dt;
    char op;

    do {
        cout << "Data (dd/mm/aaaa): ";
        lerdata(dt);
        dts.push_back(dt);
        cout << "Continuar (S/N)?";
        cin >> op;
    } while(toupper(op)!='S');

    cout << "Lista Original:\n";
    for(auto d: dts) exibedata(d);

    cout << "Ordenando pelo dia:\n";
    sort(dts.begin(), dts.end(), ordena);
    for(auto d: dts) exibedata(d);
    return 0;
}

```