Aula 01:

Introdução ao C++

ECOP13A - Programação Orientada a Objetos
Prof. André Bernardi
andrebernardi@unifei.edu.br

Universidade Federal de Itajubá

Plano de Curso

ECOP13A - Programação Orientada a Objetos – 64 h

Turmas:

T01 (2025.1 - 4T12 6T34),
T02 (2025.1 - 4T34 6T12)

- **Pré-Requisitos:** ECOP11A
- **Ementa:** Introdução à linguagem C++. Classes, objetos e abstração de dados. Sobrecarga de operadores. Herança. Funções virtuais e polimorfismo. Tratamento de exceções. Gabaritos. Introdução à biblioteca padrão de gabaritos (STL).

Plano de Curso

Objetivos:

- Introduzir os alunos ao conceito de programação orientada a objetos através da linguagem de programação C++.
- Habilitar os alunos a desenvolverem aplicações mais complexas, com hierarquia de classes e polimorfismo.
- Apresentá-los à biblioteca padrão de gabaritos (STL), tornando-a uma ferramenta para solução de problemas.

Plano de Curso

Bibliografia:

- Deitel – C++ Como Programar (várias edições)
- Bjarne Stroustrup – The C++ Programming Language Fourth Edition (2013) - ISBN 978-0-321-56384-2 – Addison-Wesley
- Notas de aula do curso.

Plano de Curso

Avaliações:

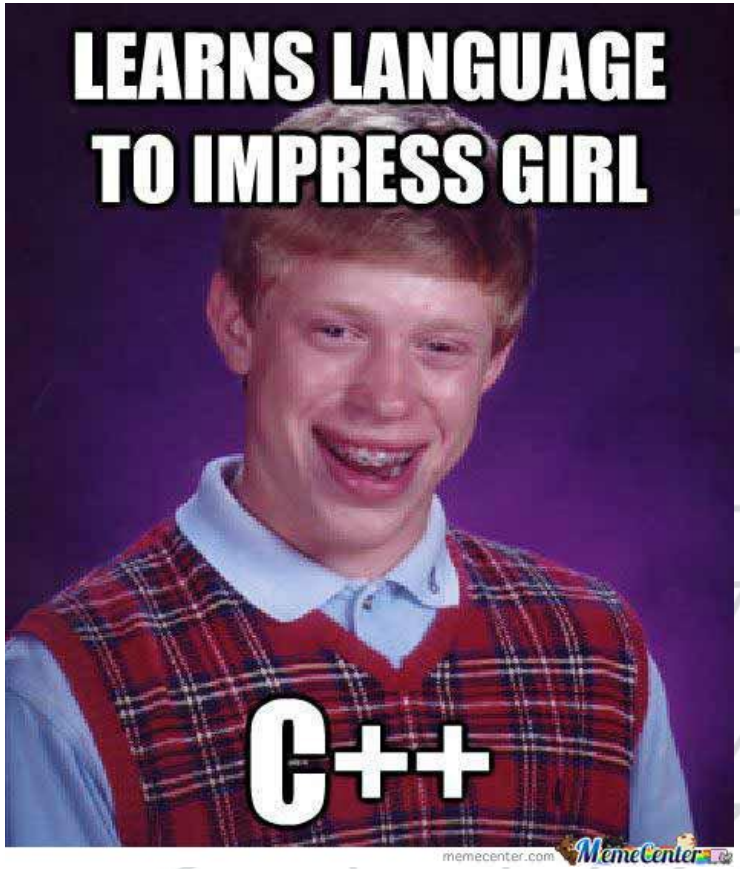
- Primeiro Bimestre:
  - Labs de 01 a 06 – 28/03 a 16/05 – 50%
  - Avaliação I – 23 de Maio – 50%
- Segundo Bimestre:
  - Labs de 07 a 12 – 30/05 a 04/07 – 50%
  - Avaliação II – 09 de julho – 50%
- Nota final: Média aritmética

Plano de Curso

Tabela de Horários para GRADUAÇÃO:

Horários	Seg	Ter	Qua	Qui	Sex	Sab
13:30 - 14:25	---	---	ECOP13A T1	---	ECOP13A T2	---
14:25 - 15:20	---	---	ECOP13A T1	---	ECOP13A T2	---
15:45 - 16:40	---	---	ECOP13A T2	---	ECOP13A T1	---
16:40 - 17:35	---	---	ECOP13A T2	---	ECOP13A T1	---


WHAT ARE YOU STUDYING ?
C++
quickmeme.com


LEARNS LANGUAGE
TO IMPRESS GIRL
C++
memecenter.com
MemeCenter

Agenda

Início	Fim	Descrição
12/03/2025	12/03/2025	Integra
14/03/2025	14/03/2025	Apresentação da disciplina - Introdução ao C++
21/03/2025	04/04/2025	Introdução ao C++
09/04/2025	16/04/2025	Classes Parte I
23/04/2025	30/04/2025	Classes Parte II
02/05/2025	02/05/2025	Recesso
07/05/2025	09/05/2025	Sobrecarga de Operadores - Parte I
14/05/2025	16/05/2025	Sobrecarga de Operadores - Parte II
21/05/2025	21/05/2025	Revisão
23/05/2025	23/05/2025	Avaliação I
28/05/2025	30/05/2025	Herança
04/06/2025	06/06/2025	Polimorfismo
11/06/2025	13/06/2025	Templates
18/06/2025	18/06/2025	Tratamento de Exceções
25/06/2025	27/06/2025	STL - Parte I
02/07/2025	04/07/2025	STL - Parte II
09/07/2025	09/07/2025	Avaliação II
11/07/2025	11/07/2025	Não Haverá Aula
16/07/2025	16/07/2025	Avaliação Substitutiva
18/07/2025	18/07/2025	Não Haverá Aula


DON'T
PANIC

Propósito de Linguagem de Programação:

“expressar ideias em código”

Duas tarefas básicas:

1. Prover um veículo para que o programador especifique ações a serem executadas pela máquina;
2. Prover um conjunto de conceitos que o programador deve pensar enquanto decide o que será feito.

A primeira tarefa requer, idealmente, uma linguagem “**próxima à máquina**”, de maneira que todos os aspectos da máquina sejam tratados com simplicidade e eficiência, de uma maneira razoavelmente óbvia para o programador. C foi projetada com este propósito.

A segunda requer uma linguagem que seja “**próxima do problema a ser resolvido**”, de maneira que os conceitos que envolvam uma solução possam ser expressos de maneira direta e concisa.

C++ acrescenta ao C uma série de funcionalidades e ferramentas visando aproximar a linguagem do problema a ser resolvido: **classes, construtores e destrutores, exceções, gabaritos, checagem de argumento em funções, etc.**

C++ melhora muitos dos recursos de C e fornece capacidades de programação orientada a objetos (OOP) que aumentam a produtividade, qualidade e **reutilização** do software.

C++, portanto, provê:

- Mapeamento direto de operações e tipos nativos para hardware, provendo uso eficiente de memória e operações de nível mais baixo;
- E mecanismos de abstração flexíveis e de “custo” acessível, para prover tipos definidos pelo usuário com o mesmo suporte de notação, variedade de utilização e desempenho dos tipos nativos.

Simula	+	C	=	**C++**

Ole-Johan Dahl
Funcionalidades para organização do programa e mecanismos de abstração

Dennis Ritchie
Eficiência e flexibilidade em programação de sistemas

Bjarne Stroustrup
Cria em 1979 a primeira implementação de “C com classes”, posterior **C++.**

Bjarne Stroustrup

Um pouco de História

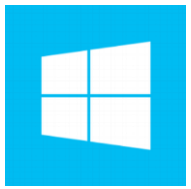
- [1979] Começa o trabalho em "***C com classes***", já incluindo classes, herança, controle de acesso público/privado, construtores e destrutores, etc.
- [1984] "C com classes" é renomeado para **C++** e ganha novas funcionalidades como funções virtuais, sobrecarga de operadores, etc.
- [1985] Primeiro uso comercial de C++ e publicação do livro "The C++ Programming Language".
- [1991] "The C++ Programming Language" 2º edição, apresentando programação com templates e tratamento de exceções.
- [1997] "The C++ Programming Language" 3º edição, introduzindo ISO C++ e STL.
- [1998] Padrão ISO C++.
- [2011] **Padrão ISO C++ 11** formalmente aprovado, com adição de novas características como inicialização uniforme, *move*, expressões lambda, modelo de memória apropriado para concorrência, auto, range for, etc. Utilizaremos esta versão.

Onde C++ é utilizado?




C++ é utilizado praticamente em TODO lugar. Normalmente você não o vê: C++ é uma linguagem de programação de sistemas e sua utilização se dá em grande escala na parte de **infraestrutura**, onde ninguém geralmente olha.

Por exemplo, muitos dos sistemas operacionais atuais possuem grandes partes escritas em C++: Windows, Mac OS, Linux. Além disso, roteadores de internet, *device drivers*, e qualquer software projetado para utilizar o máximo do hardware.



Onde mais C++ é utilizado?

A maioria dos mais utilizados e conhecidos sistemas atuais possuem suas partes mais críticas escritas em C++:

Além disso, muitas outras linguagens dependem de C++ para sua implementação:

- Java Virtual Machines (Oracle Hotspot)
- Javascript (Google V8)
- Navegadores de internet (Firefox, Safari, Chrome, Opera)
- Frameworks de aplicação (Microsoft .NET)

Onde mais o C++ é utilizado?

Suas características de escalabilidade, estabilidade e compatibilidade o levaram até:

- Mercado Financeiro
- Telecomunicações (todas as ligações de longa distância nos EUA são roteadas por programas em C++)
- Aplicações militares
- Games!
- Sistemas Embarcados
  - Tomografia Computadorizada (CAT Scanners)
  - Controle de Voo (Lockhead-Martin)
  - Controle de foguetes
  - Softwares automobilísticos (BMW)
  - Controle de turbinas (Vesta)
- Aplicações científicas
  - Projeto Genoma Humano
  - NASA Mars Rovers
  - CERN
- Aplicações gráficas

Etc., etc., etc.

Interoperabilidade e Bibliotecas

C++ é projetado de maneira que seu código possa coexistir com códigos escritos em outras linguagens. **Grandes sistemas não são escritos em apenas uma linguagem**, e o projeto de C++ com foco em interoperabilidade torna-se ainda mais importante.

Sistemas grandes dificilmente são escritos apenas com as funcionalidades nativas da linguagem. C++ é apoiado por uma grande variedade de bibliotecas (além da padrão!) para vários fins:

boost C++ LIBRARIES

Vários fins

POCO C++ LIBRARIES

Desenvolvimento Web

Qt The Qt Company

Desenvolvimento de aplicações *cross-platform*

CGAL Computational Geometry Algorithms Library

wxWidgets Cross-Platform GUI Library

ROOT Data Analysis Framework

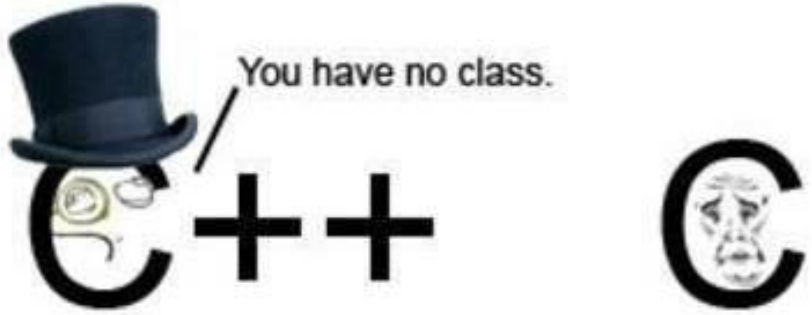
Aplicações científicas (CERN)

OpenCV

Processamento de imagens

Conselhos para o programador de C:

- Não pense em C++ como a própria linguagem C com algumas novas funcionalidades. C++ pode ser utilizado desta maneira, mas nunca de maneira ótima. Para aproveitar todo o poder de C++ aprenda o novo estilo de implementação e projeto.
- Não escreva código típico de C em C++. **Minimize** a utilização de strings e vetores em estilo de C, uso de malloc e *free*, aritmética de ponteiros, etc.
- Prefira sempre o funcionalidades da biblioteca padrão a código escrito localmente. Aprenda e utilize a biblioteca padrão de C++ sempre que possível. Não pense que um código laborioso escrito na raça em C será melhor que alguma funcionalidade da biblioteca padrão de C++. Normalmente o contrário é verdadeiro. **Reutilização**

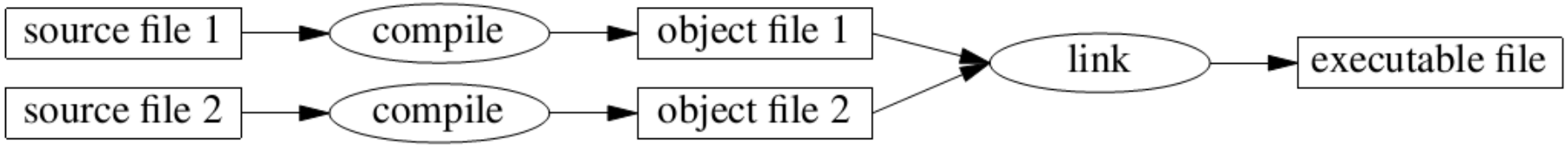


A C++ Basic Tour

Passeando pela linguagem

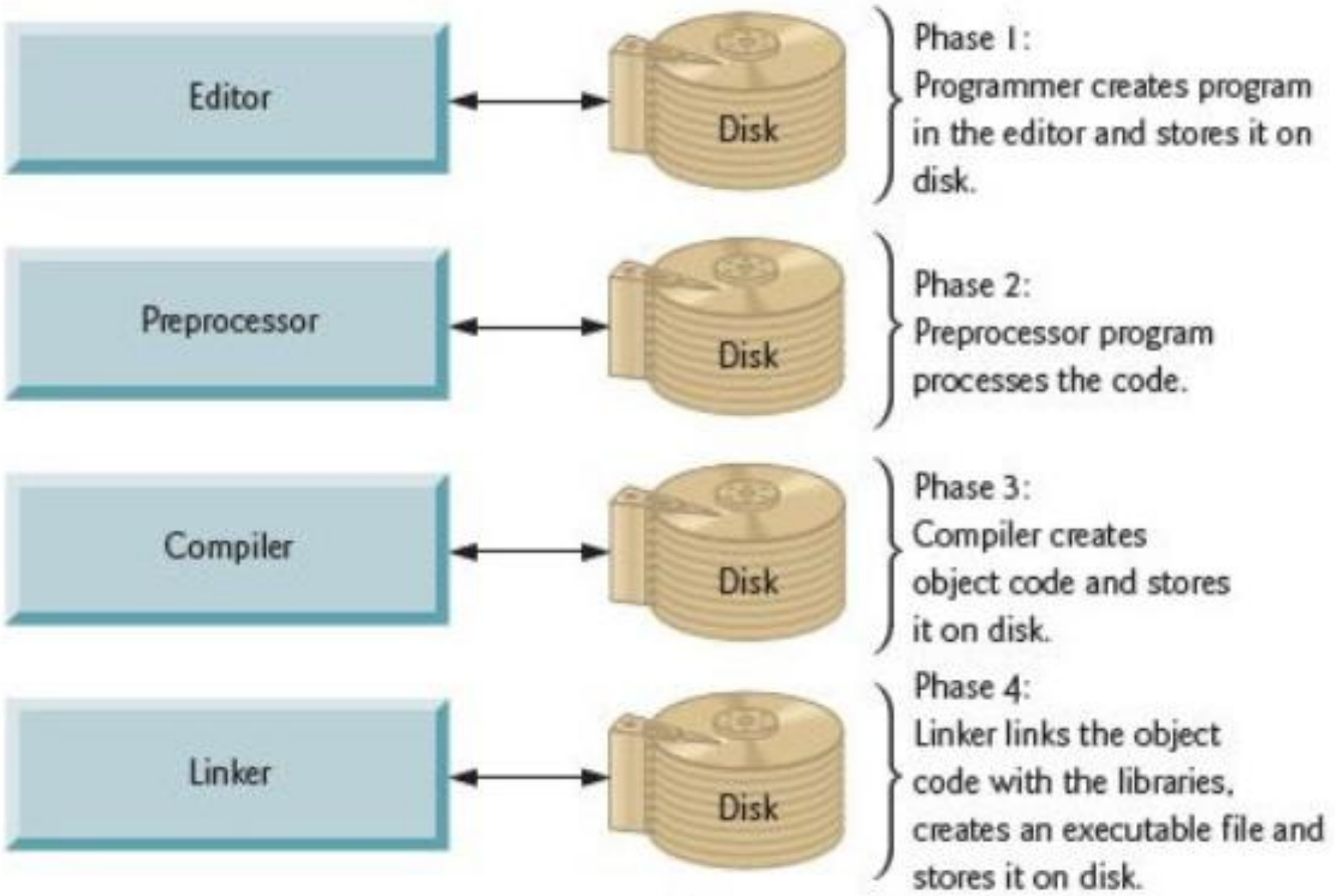
C++ é uma linguagem compilada.

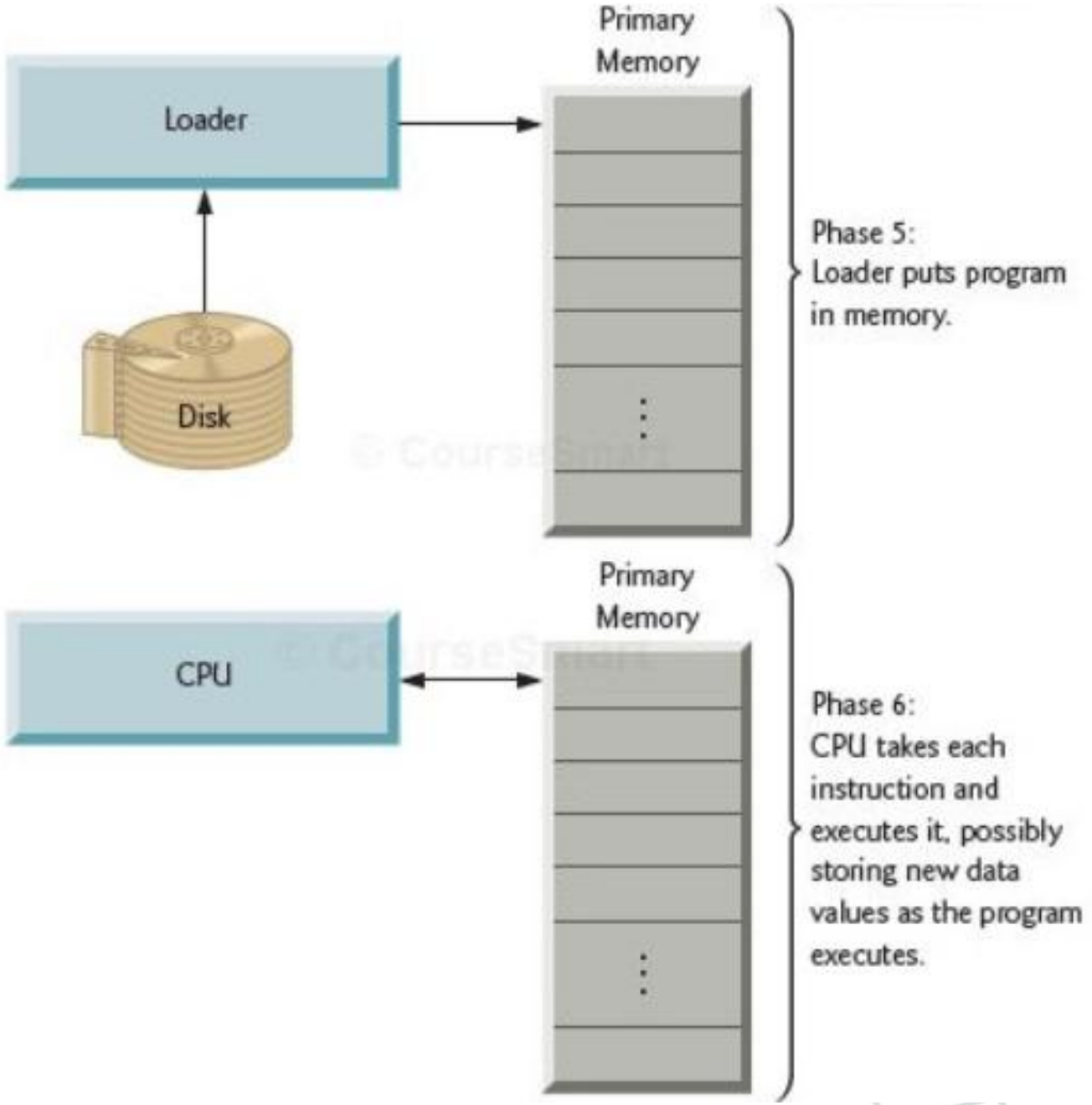
Para que um programa seja executado, seus arquivos de código-fonte devem ser processados por um **compilador**, produzindo arquivos objeto que são combinados por um ***linker*** em um programa executável.



Um programa executável é criado para uma combinação específica de hardware e sistema e, por exemplo, não é portável de um Mac para um PC com Windows.

Quando falamos em portabilidade em C++, estamos falando de portabilidade de código-fonte, que pode ser compilado e executado em uma grande variedade de sistemas sem modificação alguma.


Editor
Disk
Phase 1:
Programmer creates program in the editor and stores it on disk.
Preprocessor
Disk
Phase 2:
Preprocessor program processes the code.
Compiler
Disk
Phase 3:
Compiler creates object code and stores it on disk.
Linker
Disk
Phase 4:
Linker links the object code with the libraries, creates an executable file and stores it on disk.


Primary
Memory
Loader
Disk
Phase 5:
Loader puts program
in memory.
Primary
Memory
CPU
Phase 6:
CPU takes each
instruction and
executes it, possibly
storing new data
values as the program
executes.

O padrão ISO C++ define dois tipos diferentes de entidades:

- *Core language features*: são os tipos nativos (como char e int) e operações básicas como laços (for, while)
- Componentes da biblioteca padrão: são componentes também presentes em todas as implementações de C++, mas que são implementados na própria linguagem, com a finalidade de prover mais funcionalidades de alto nível. Como exemplos temos os containers (vector, map) e operações de entrada e saída.

C++ é uma linguagem estaticamente “tipada”, o que significa que cada entidade (objeto, valor, nome e expressão) precisa ser conhecida pelo compilador em seu ponto de utilização. O tipo de uma variável (ou objeto) determina o conjunto de operações aplicáveis a ela.
Vejamos um exemplo de código:

```cpp
#include <iostream>

int squared(int n)
{
    return n*n;
}

int main()
{
    std::cout << "Por favor, entre com um numero inteiro: ";

    int number = 0;
    std::cin >> number;

    std::cout << number << "^2 = " << squared(number) << std::endl;
}
```

Veja como o código é parecidíssimo com o C.

```
Por favor, entre com um numero inteiro: 10
10^2 = 100
```

- Todo programa em C++ deve possuir uma (e somente uma) função main.
- A definição e chamada da função são similares ao que aprendemos em C.
  - Repare que a função está sendo declarada e definida antes de sua utilização. Uma alternativa seria a utilização de um protótipo.

```cpp
#include <iostream>

int squared(int n)
{
    return n*n;
}

int main()
{
    std::cout << "Por favor, entre com um numero inteiro: ";

    int number = 0;
    std::cin >> number;

    std::cout << number << "^2 = " << squared(number) << std::endl;
}
```

```
Por favor, entre com um numero inteiro: 10
10^2 = 100
```

- A diretiva #include é a mesma, e serve para indicar quais bibliotecas iremos utilizar em nosso código, mas agora utilizaremos outra biblioteca para entrada e saída padrão de dados, a ***iostream***.
- O **std::** indica que os comandos *cout*, *cin* e *endl* podem ser encontrados no namespace da biblioteca padrão. Veremos uma maneira para que não seja necessário escrevê-lo todas as vezes.

```cpp
#include <iostream>

int squared(int n)
{
    return n*n;
}

int main()
{
    std::cout << "Por favor, entre com um numero inteiro: ";

    int number = 0;
    std::cin >> number;

    std::cout << number << "^2 = " << squared(number) << std::endl;
}
```

Saídas de Stream Concatenadas

- A execução da última linha produz valores de diferentes tipos. O operador de inserção de fluxo “sabe” como produzir cada tipo de dado. Usar vários operadores de inserção de fluxo (<<) em uma única declaração é chamado de **concatenação**, **encadeamento** ou **operações de inserção de fluxo em cascata**.
- Cálculos e chamada de funções também podem ser realizados nesta operação.

```
Por favor, entre com um numero inteiro: 10
10^2 = 100
```

Outro Exemplo:

```cpp
#include <iostream>

// declara que torna acessível o namespace std sem qualificação (std::)
using namespace std;

// protótipo da função
int cube(int);

int main()
{
    cout << "Entre com um numero inteiro: "; // Não é necessário std::

    int number = 0;
    cin >> number;

    cout << "Resultado: " << number << "^3 = " << cube(number) << endl;
}

// definição da função
int cube(int n)
{
    return n*n*n;
}
```

```
Entre com um numero inteiro: 10
Resultado: 10^3 = 1000
```

Declaração `return` não obrigatória na main:

Note que não há uma declaração **`return 0;`** no final da `main` neste exemplo. De acordo com o padrão C++, se a execução do programa atingir o final de **`main`** sem encontrar uma declaração **`return`**, presume-se que o programa foi encerrado com sucesso — exatamente como quando a última instrução na **`main`** é uma declaração **`return`** com o valor **`0`**. Portanto, podemos omitir a declaração **`return`** no final de **`main`** nos programas em C++.

Arquivos de Cabeçalho

A **Biblioteca Padrão C++** é dividida em muitas partes, cada uma com seu próprio arquivo de cabeçalho. Os arquivos de cabeçalho contêm os **protótipos de função** para as funções relacionadas que formam cada parte da biblioteca. Os arquivos de cabeçalho também contêm definições de vários tipos de **classe e funções**, bem como **constantes** necessárias para essas funções. Um arquivo de cabeçalho "instrui" o compilador sobre como interagir com a biblioteca e os componentes escritos pelo usuário.

Os nomes de arquivos de cabeçalho terminados em **`.h`** são arquivos de cabeçalho no "*formato antigo*" que foram substituídos pelos arquivos de cabeçalho da Biblioteca Padrão C++.

C++ Standard Library header file

<iostream>
<iomanip>
<cmath>
<cstdlib>
<ctime>
<array>,
<vector>, <list>,
<forward_list>,
<deque>, <queue>,
<stack>, <map>,
<set>, <bitset>

<cstring>
<typeinfo>
<exception>,
<stdexcept>
<memory>
<fstream>
<string>
<sstream>
<functional>
<iterator>

<algorithm>
<cassert>
<cfloat>
<climits>
<cstdio>
<locale>
<limits>
<utility>

Arquivos de Cabeçalho

Arquivos de cabeçalho personalizados, definidos pelo programador, devem terminar em **`.h`**. Um arquivo de cabeçalho definido pelo programador pode ser incluído usando a diretiva de pré-processador **`#include`**. Por exemplo, o arquivo de cabeçalho `square.h` pode ser incluído em um programa colocando a diretiva **`#include "square.h"`** no início do programa.

Todo identificador ou expressão presente no código possui um tipo, que determina as operações que podem ser realizadas nela e seu intervalo de valores.

Por exemplo:

```
int polegada = 2;
```

Especifica que "*polegada*" é do tipo `int`, ou seja, é um inteiro.

Declaração é a instrução que apresenta um nome ao programa. Nela, especificamos um tipo para uma entidade nomeada.

Declarações podem ser colocadas em quase qualquer lugar em um programa C++, mas elas devem aparecer antes que suas variáveis correspondentes sejam usadas no programa.

Uma **variável** deve, sempre que possível, ser **inicializada** no ato de sua **declaração**, portanto, não introduza um novo nome até que você tenha um valor adequado para ele.

C++ nos oferece algumas maneiras diferentes de se expressar a inicialização, como o **operador =**, que já conhecemos, e através de **listas delimitadas por chaves**. Veja:

```
double d1 = 2.3;          // inicializa d1 com o valor 2.3
double d2 {2.3};          // inicializa d2 com o valor 2.3
double d3 = {2.3};        // inicializa d3 com o valor 2.3 (= é redundante)

int v[ ] {1, 2, 3, 4, 5}; // cria um vetor de inteiros com cinco valores
```

O **operador =** é tradicional e é herdado do C. No entanto, o uso do **inicializador em chaves** é mais forte na **checagem de tipo**, não permitindo atribuições com perda de informação. Veja:

```
int i1 = 7.2;   // i1 recebe o valor 7 (estava esperando por isso?)
int i2 {7.2};   // erro: tentativa de conversão de número real para inteiro
```

Ao definimos uma variável, não é necessário declarar explicitamente seu tipo, se ele puder ser deduzido do inicializador. Por exemplo:

```
auto b = true;      // bool
auto ch = ‘x’;      // char
auto i = 123;       // int
auto d = 1.2;       // double
auto z = sqrt(y);   // o tipo de retorno de sqrt
```

Quando utilizamos **auto**, é preciso inicializar com o operador **=** pois, como não há conversão de tipos envolvida, alguns problemas podem surgir.

Este tipo de inicialização é especialmente interessante quando estamos trabalhando com programação genérica, onde os nomes de tipos são realmente grandes. Vamos chegar lá aqui a alguns meses. Enquanto isso, podemos utilizá-lo sempre que não haja uma razão específica para menção explícita do tipo.

Operadores Aritméticos

C++ operation	C++ arithmetic operator	Algebraic expression	C++ expression
Addition	+	$f + 7$	f + 7
Subtraction	-	$p - c$	p - c
Multiplication	*	bm or $b \cdot m$	b * m
Division	/	x / y or $\frac{x}{y}$ or $x \div y$	x / y
Modulus	%	$r \bmod s$	r % s

Fig. 2.9 | Arithmetic operators.

Fig. 2.10 Precedence of arithmetic operators.

Operator(s)	Operation(s)	Order of evaluation (precedence)
`( )`	Parentheses	Evaluated first. For *nested* parentheses, such as in the expression `a * (b + c / (d + e))`, the expression in the *innermost* pair evaluates first. [*Caution:* If you have an expression such as `(a + b) * (c - d)` in which two sets of parentheses are not nested, but appear "on the same level," the C++ Standard does *not* specify the order in which these parenthesized subexpressions will evaluate.]

`*`
`/`
`%`
	Multiplication
	Division
	Remainder
		Evaluated second. If there are several, they're evaluated left to right.

`+` `-`	Addition Subtraction	Evaluated last. If there are several, they're evaluated left to right.

Operadores Relacionais

Standard algebraic equality or relational operator	C++ equality or relational operator	Sample C++ condition	Meaning of C++ condition
Relational operators			
>	>	x > y	x is greater than y
<	<	x < y	x is less than y
≥	>=	x >= y	x is greater than or equal to y
≤	<=	x <= y	x is less than or equal to y
Equality operators			
=	==	x == y	x is equal to y
≠	!=	x != y	x is not equal to y

Fig. 2.12 | Equality and relational operators.

Fig. 2.14 Precedence and associativity of the operators discussed so far.

Operators				Associativity	Type
`()`				*[See caution in* **Fig. 2.10***]*	grouping parentheses
`*`	`/`	`%`		left to right	multiplicative
`+`	`-`			left to right	additive
`<<`	`>>`			left to right	stream insertion/extraction
`<`	`<=`	`>`	`>=`	left to right	relational
`==`	`!=`			left to right	equality
`=`				right to left	assignment

Operators						Associativity	Type
`::`	`()`					left to right *[See **Fig. 2.10**'s caution regarding grouping parentheses.]*	primary
`++`	`--`	`static_cast<type>()`				left to right	postfix
`++`	`--`	`+`	`-`			right to left	unary (prefix)
`*`	`/`	`%`				left to right	multiplicative
`+`	`-`					left to right	additive
`<<`	`>>`					left to right	insertion/extraction
`<`	`<=`	`>`	`>=`			left to right	relational
`==`	`!=`					left to right	equality
`?:`						right to left	conditional
`=`	`+=`	`-=`	`*=`	`/=`	`%=`	right to left	assignment

C++ Keywords

Keywords common to the C and C++ programming languages

auto	break	case	char	const
continue	default	do	double	else
enum	extern	float	for	goto
if	int	long	register	return
short	signed	sizeof	static	struct
switch	typedef	union	unsigned	void
volatile	while			

C++-only keywords

and	and_eq	asm	bitand	bitor
bool	catch	class	compl	const_cast
delete	dynamic_cast	explicit	export	false
friend	inline	mutable	namespace	new
not	not_eq	operator	or	or_eq
private	protected	public	reinterpret_cast	static_cast
template	this	throw	true	try
typeid	typename	using	virtual	wchar_t
xor	xor_eq			

Fig. 4.3 | C++ keywords.

Estruturas de Controle

C++ provê um conjunto de instruções convencionais de seleção e repetição, da mesma maneira que estudamos em **C**:

Seleção:
- if
- if/else
- switch

Repetição:
- while
- do/while
- for
- range for (novo!)

Dicas:
- Prefira um switch ao if, onde houver escolha.
- Prefira um for a um while sempre que houver uma variável de controle óbvia.
- Prefira um while a um for sempre que NÃO houver uma variável de controle óbvia.
- Evite instruções do/while. O fato de o código executar uma vez sem que a condição seja testada pode ser fonte de erros e confusões.

```cpp
#include <iostream>

using namespace std;

bool accept()
{
    int tentativas = 0;

    while(tentativas < 4)
    {
        cout << "Voce deseja continuar? (y/n) ";

        char resp = 0;
        cin >> resp;

        switch(resp)
        {
            case 'y':
            case 'Y': return true;
            case 'n':
            case 'N':
                    return false;
            default:
                cout << "Resposta invalida." << endl;
                tentativas++;
        }
    }

    cout << "Bom, eu acho que isso foi um nao." << endl;
    return false;
}
```

Repare que as instruções **if/else**, **switch** e **while** ainda funcionam exatamente da maneira como você se lembra (ou não).

```cpp
int main()
{
    if(accept())
        cout << "Voce continua!" << endl;
    else
        cout << "Voce parou..." << endl;
}
```

Com relação à instrução **for**, temos algumas diferenças e uma nova opção.

```cpp
#include <iostream>

using namespace std;

int main()
{
    int vetor[] {1, 2, 3, 4, 5, 6, 7, 8, 9};

    cout << "Percorrendo com for tradicional: ";
    for(int i = 0; i < 9; i++)
        cout << vetor[i] << " ";

    cout << endl;

    cout << "Percorrendo com range for: ";
    for(int x: vetor) // ou ainda for(auto x: vetor)
    {
        cout << x << " ";
    }
}
```

for

O primeiro é parecido com o que já conhecemos, com a diferença que agora podemos realizar a declaração da variável de controle dentro da própria instrução, como int i = 0 no exemplo.

range for

Essa instrução pode ser lida da seguinte maneira: para cada inteiro x no vetor “vetor”, faça o que está entre chaves. É equivalente à instrução foreach de algumas outras linguagens (C#)

A biblioteca padrão oferece apenas entrada e saída de caracteres formatados, através da biblioteca **iostream**. Qualquer outra forma de interação com o usuário, com I/O gráfica, é realizada através de outras bibliotecas não presentes no padrão ISO.

Saída de Caracteres

O operador << (“colocar em”) é utilizado como operador de saída em objetos do tipo ***ostream***, sendo que cout é o stream de saída padrão e cerr é o stream padrão para relato de erros. Por exemplo, podemos escrever:

```
void f()
{
    cout << 10; // coloque o literal 10 no stream padrão de saída

    int i = 10;
    cout << i; // coloque o conteúdo da variável i na saída

    // caso ocorra algum erro, podemos escrever no stream de erros
    cerr << "Ocorreu um erro no seu programa";
}
```

É possível ainda, combinar uma sentença inteira em uma única linha:

```
int idade {30};
cout << "O professor tem " << idade << " anos\n";
```

```
O professor tem 30 anos
```

Entrada de Caracteres

O **operador >>** (“coletar de”) é utilizado como operador de entrada em objetos do tipo ***istream***, sendo que **cin** é o stream de entrada padrão. Por exemplo, podemos escrever:

```
void g()
{
    // entrando com um unico numero
    cout << "Entre com um inteiro: ";
    int i = 0;
    cin >> i;

    // entrando com dois numeros, separados por espaço ou enter
    cout << "Entre com dois doubles: ";
    double a = 0.0, b = 0.0;
    cin >> a >> b;
}
```

Com frequência, precisaremos ler valores textuais. Para isso, podemos utilizar o tipo string, do C++.

```
#include <string>

void h()
{
    cout << "Entre com seu nome: ";
    string str;
    cin >> str;
    cout << "Oi, " << str << "!\n";
}
```

Dica:

- Iremos sempre preferir a implementação de **string** da biblioteca padrão de C++ à implementação em vetor de char de C.

Entrada de Caracteres

Por padrão, um caractere de espaço em branco termina a leitura da entrada. Portanto, caso o nome seja “Joao Paulo”, somente “Joao” seria lido.

Caso queira ler uma linha inteira de texto (incluindo o caractere de terminação de linha), há outra função que deve ser utilizada: **getline()**.

```
void h()
{
    cout << "Entre com seu nome: ";
    string str;
    getline(cin, str); // argumentos: stream de saída cin e a string a armazenar valor
    cout << "Oi, " << str << "!\n";
}
```

```
Entre com seu nome: Joao Paulo
Oi, Joao Paulo!
```

Em um exemplo anterior, foi utilizado o comando **std::endl** ao final da linha da seguinte maneira:

```
cout << "Voce parou..." << endl;
```

Dicas:
Este comando também é adequado para terminar a linha. No entanto, além de acrescentar o final da linha, ele também realiza um **std::flush** na saída. Essa operação custa tempo. Portanto, sempre que possível prefira **‘\n’** a **endl**.

Processamento de strings

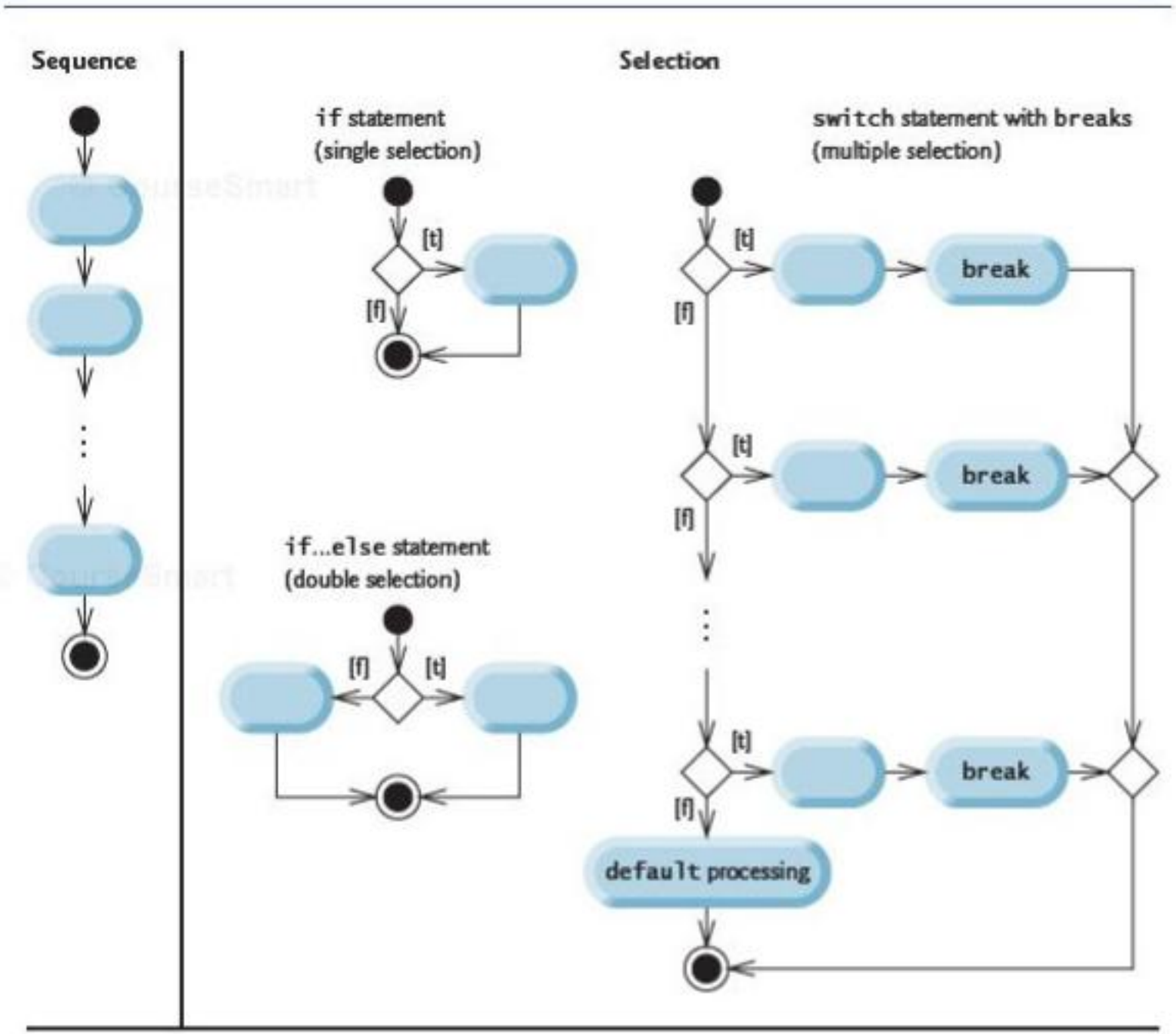
```cpp
#include <iostream>
#include <string>

using namespace std;

int main()
{
    cout << "Entre com seu nome: ";

    string nome;
    cin >> nome;

    cout << "Agora entre com seu sobrenome: ";
    string sobre;
    cin >> sobre;

    // concatenando no novo
    string nome_completo = nome + " " + sobre;
    cout << "Nome completo: " << nome_completo << "\n";

    // concatenando no primeiro
    nome += " ";
    nome += sobre;
    cout << "Nome completo: " << nome << "\n";

    // comparando com um literal
    if(nome == "Joao Paulo")
        cout << "Ta tranquilo. Ta favoravel. \n";
}
```

Repare como é muito mais simples do que o que fazíamos em C. Atribuição pode ser realizada diretamente, utilizando o operador =, e comparações de igualdade podem ser feitas com ==.

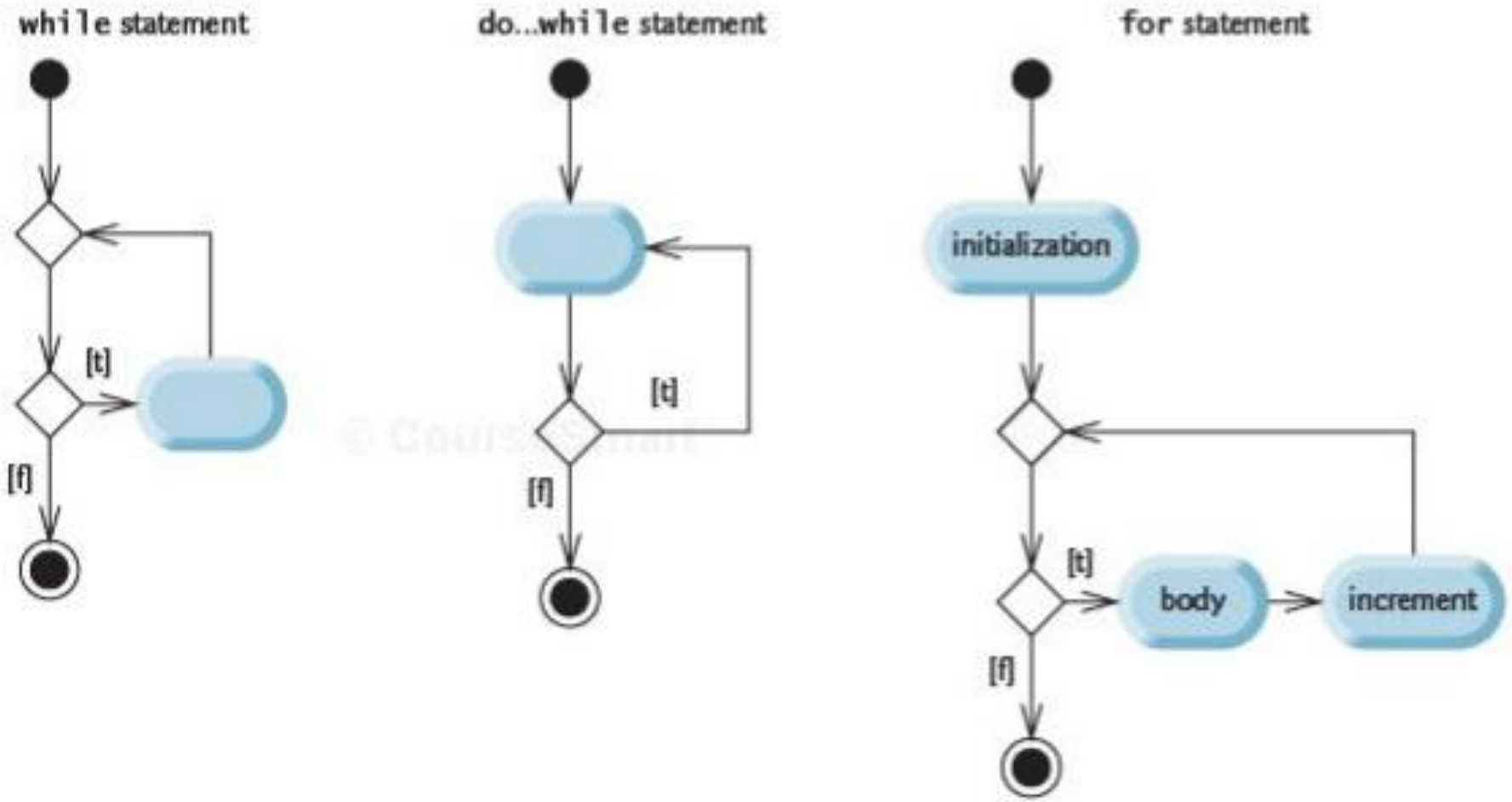
Não é necessário utilizar strcmp, strcpy, etc.

Dicas: A biblioteca **string** é muito rica. Estude sua referência na internet e aprenda do que mais ela é capaz.

```
Entre com seu nome: Joao
Agora entre com seu sobrenome: Paulo
Nome completo: Joao Paulo
Nome completo: Joao Paulo
Ta tranquilo. Ta favoravel.
```


Sequence
Selection
if statement
(single selection)
[t]
[f]
switch statement with breaks
(multiple selection)
[t]
[f]
break
[t]
[f]
break
if...else statement
(double selection)
[f]
[t]
[t]
[f]
break
default processing

Repetition


while statement
[t]
[f]
do...while statement
[t]
[f]
for statement
initialization
[t]
body
increment
[f]

Referências

- https://cplusplus.com/reference/

- Notas de aula da disciplina Programação Orientada a Objetos, Prof. André Bernardi, Prof. João Paulo Reus Rodrigues Leite.