Aula 1

- C é uma linguagem de programação estruturada desenvolvida por Dennis Ritchie nos laboratórios Bell entre 1969 e 1972;
- **Algumas características**:
  - É *case-sensitive*, ou seja, o compilador difere letras maiúsculas e minúsculas;
  - Tem poucas palavras reservadas;
  - Tem bom controle do hardware;
- Lembrando que há dois tipos de erros (sempre provocados pelo próprio programador):
  - Lógico: não pode ser detectado em tempo de compilação;
  - Sintaxe : a identificação é possível durante a compilação do programa.
- A linguagem C++ foi desenvolvida por Bjarne Stroustrup em 1983 nos laboratórios Bell como um adicional à linguagem C. Novas características foram adicionadas com o tempo, como funções virtuais, sobrecarga de operadores, herança múltipla e tratamento de exceções. Desde que a linguagem C++ faz um melhor tratamento de strings, faremos nossos programas em C e somente a parte de strings usando uma biblioteca C++. Consequentemente, compilaremos os programas em C++.

Primeiro programa: um Hello World

- **Exercício 1.** Vamos fazer um "*hello world*!" em C++:

```
// exemplo1.cpp
/* Incluímos a biblioteca C++ padrão de entrada e saída */
#include <iostream>
main(){
      std::cout << "Hello World!" << std::endl;
}
```

ou equivalentemente:

```
// exemplo1.cpp
/* Incluímos a biblioteca C++ padrão de entrada e saída */
#include <iostream>
using namespace std;

main(){
      cout << "Hello World!" << endl;
}
```

`std::cout` é a saída padrão e `std::endl` pula linha.

Para compilar o programa em um terminal no Linux, digite:

g++ -o exemplo1 exemplo1.cpp

Para executar o programa, digite:

./exemplo1

No Windows, você pode utilizar o Dev-C++ instalado nos laboratórios da UFABC.

Comparando com comandos Java vistos em Processamento da Informação

	C	Java
Atribuição	=	=
Operadores Aritméticos	+, -, *, /, %	+, -, *, /, %
Operadores Aritméticos Unários	++, --	++, --
Operadores Relacionais	<, >, <=, >=, ==, !=	<, >, <=, >=, ==, !=
Operadores Lógicos	&&, ||, !, (… ? … : …)	&&, ||, !, (… ? … : …)
Imprimir na tela	cout << "...";	System.out.println();
Ler dados do usuário	cin >> variavel cin.getline (var_string, length)	Scanner sc= new Scanner(System.in); a = sc.nextDouble();
Laço	for (i= ..;...;)	for (i= ..;...;)
Delimitação de um bloco	{ }	{ }
Laço	while(...) { }	while(...) { }
Laço	do { } while(...)	do { } while(...)
Comentário de várias linhas	/* ... */	/* ... */
Comentário de uma linha	//	//
Condicional	if (condição) { ... } else { ... }	if (condição) { ... } else { ... }
Condicional	switch (expressão){case 0: ... }	switch (expressão) {case 0: ... }
Retorno	return	return
Declaração de constante	const float Pi=3.1416;	final float Pi=3.1416;

Tipos de Dados Primitivos mais Usuais

Tipo	Tamanho em Bytes	Faixa Mínima
char	1	-127 a 127
unsigned char	1	0 a 255
signed char	1	-127 a 127
int	4	-2.147.483.648 a 2.147.483.647
unsigned int	4	0 a 4.294.967.295
signed int	4	-2.147.483.648 a 2.147.483.647
short int	2	-32.768 a 32.767
unsigned short int	2	0 a 65.535
signed short int	2	-32.768 a 32.767
long int	4	-2.147.483.648 a 2.147.483.647
signed long int	4	-2.147.483.648 a 2.147.483.647
unsigned long int	4	0 a 4.294.967.295
float	4	Seis dígitos de precisão
double	8	Dez dígitos de precisão
long double	10	Dez dígitos de precisão

- Todos os tipos inteiros são `signed` por *default*;

Declaração de Variáveis

Forma geral:

```
tipo variavel;
```

Alguns exemplos:

```
int i=0;
unsigned int x,y,z;
unsigned short int idade;
float j = 2.0;
double nota_disciplina;
int v[100]; // vetor vai de 0 a 99
int v[100] = {0}; // inicializa vetor com 0
char nome[4] = {'a', 'b', 'a', '\0'}; // equivale a String em Java
char nome[] = "Estrutura de dados"; // equivale a String em Java
```

- Todos as variáveis devem ser inicializadas;

Imprimindo e lendo dados do usuário

- **Para ler dados do teclado:** utilizamos a função `std::cin.getline(variavel, length_var)` para ler dados do tipo `String`. Para ler os demais tipos de dados, utilizamos `std::cin >> variavel`.
- **Exercício 2.** Leia o nome e a idade do usuário e imprima depois.

```
// exemplo2.cpp: leitura e escrita de dados ao usuário
#include <iostream>
using namespace std;

main(){
	int idade;
	char nome[100];
	cout << "Entre com seu nome: ";
	cin.getline(nome,100);
	cout << "Entre com sua idade: ";
	cin >> idade;
	cout << "Nome: " << nome << " Idade: " << idade << endl;
}
```

- **Observação:** atenção ao usar `std::cin.getline(variavel, length_var)` após um comando `std::cin >> variavel` . Neste caso, utilize o comando `std::cin.ignore()` antes do comando `std::cin.getline(variavel, length_var)`. O comando `std::cin.ignore()` ignora o resto da linha lida em `std::cin >> variavel`.
- **Exercício 2a.** O que está errado no código abaixo?

```
// exemplo2a.cpp: leitura e escrita de dados ao usuário
#include <iostream>
using namespace std;

main(){
	int idade;
	char nome[100];
	char apelido[100];
	cout << "Entre com seu nome: ";
	cin.getline(nome,100);
	cout << "Entre com sua idade: ";
	cin >> idade;
	cout << "Entre com seu apelido: ";
	cin.getline(apelido,100);
	cout << "Nome: " << nome << " Idade: " << idade << "Apelido: " <<
```

```
apelido << endl;
	}
```

Condicionais

- Uma condicional com uma única ação não necessita de { } para delimitar o bloco. Exemplo:

```
// exemplo3.cpp: exemplo de condicional
#include <iostream>
using namespace std;

main(){
	int x=10;
	int y=20;
	int z=30;
	// operador lógico AND
	if (x<y && z>y) cout << "z é o maior de todos!" << endl;
	// operador lógico OR
	if (x==10 || x==5) cout << "x é igual a 5 ou 10!" << endl;
}
```

- **Exercício 3.** Leia três números inteiros e ordene-os em ordem crescente. Atenção ao caso de números iguais!

Laços

- **Exemplo**:

```
// exemplo5.cpp: laços
#include <iostream>
using namespace std;

main(){
	cout << "While-do: " << endl;
	int cont=0;
	while (cont<5){
		cout << "*";
		cont++;
	}

	cout << endl << "Do-while: " << endl;
	cont=0;
	do {
		cout << "*";
		cont++;
	} while (cont<5);

	cout << endl << "For incrementando: " << endl;
	for (cont=0;cont<=5;cont++){
		cout << "Iteração número: " << cont << endl;
	}


	cout << endl << "For decrementando: " << endl;
	for (cont=5;cont>=0;cont--){
		cout << "Iteração número: " << cont << endl;
	}
}
```

Registros (Structs)

- Podemos criar tipos de dados compostos usando o comando `struct`. Exemplo:

```
// exemplo6.cpp: struct
#include <iostream>
#include <string.h>
using namespace std;

// construção de um tipo de registro
struct tipo_aluno {
	char nome[50];
	int idade;
	int ra;
	float nota;
};

main(){
	struct tipo_aluno aluno; // declarando a variável do tipo construído

	strcpy(aluno.nome,"Leticia");
	aluno.idade=20;
	aluno.ra=123456;
	aluno.nota=8.5;

	cout << "Nome " << aluno.nome << " Idade: " << aluno.idade << " RA: " << aluno.ra << " Nota: " << aluno.nota  << endl;
}
```

- Exercício 4: Modifique o programa acima, criando um vetor do tipo `tipo_aluno`. O programa deve ler um inteiro `x` e depois ler os dados de `x` registros no vetor. Ao final, o programa deve imprimir os registros armazenados no vetor.