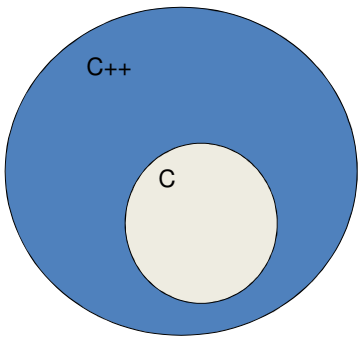
Introdução à Linguagem C++

C++: Definição



- A Linguagem de Programação C++ pode ser definida como uma extensão da Linguagem C;
- Todo código de programação em Linguagem C pode a priori ser compilado com um compilador C++;
- Stroustrup teve como principal objetivo apresentar uma linguagem de programação que mantivesse compatibilidade com C, mas que desse suporte e recursos para desenvolvimento de códigos que suportassem o paradigma da programação orientada a objetos
- O nome original sugerido por seu criador, Bjarne Stroustrup, para a sua linguagem foi "***C with Classes***"

C & C++

- C foi escolhida para ser a linguagem base para C++ pelos seguintes motivos:
  - Versátil, concisa e relativamente de baixo nível;
  - Adequado para a maioria das tarefas de programação;
  - Portável;
  - Totalmente ambientada ao UNIX;

Programação Orientada a Objetos

- C++: linguagem de programação
- POO: programação orientada a objetos
- POO é uma filosofia de programação, enquanto C++ é uma das várias linguagens de programação a apresentar recursos que possibilitem o desenvolvimento de códigos de programação com o paradigma da programação orientada a objetos;

Tópicos

- Conceitos Fundamentais
  - C e C++;
  - POO e C++;
- Expressões
  - Variáveis;
  - Operadores;
- Controle de Fluxo
  - Tomada de decisão;
  - Construção com laços;
  - Seleção;

Linguagem C++: Hello World !

```
#include <iostream>

using namespace std;

int main( void )
{
  cout << "Hello World !!!" << endl;
  return 0;
}
```

Expressões

- Na Linguagem de Programação C++, uma expressão é uma combinação de variáveis, constantes e operadores, que pode ser avaliada computacionalmente, sempre resultando em um valor (*valor da expressão*);

Variáveis

- Uma variável representa um espaço na memória do computador para armazenar um determinado tipo de dado; Em C++ todas as variáveis devem ser explicitamente declaradas;
- Na declaração da variável devem ser explicitadas sempre o *tipo* (formato do dado a ser armazenado) e o *nome* (referência para acesso);
- Só é possível armazenar valores do tipo especificado na declaração;

Tipos Básicos

- C++ oferece alguns tipos básicos:

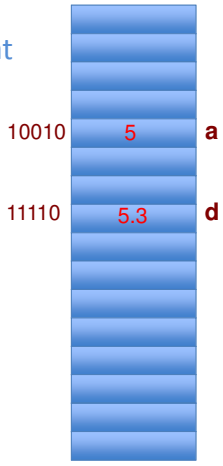
bool	*1 bit*	0, 1
char	*1 byte*	-128 a 127
unsigned char	*1 byte*	0 a 255
short int	*2 bytes*	-32768 a 32767
unsigned short int	*2 bytes*	0 a 65535
long int (int)	*4 bytes*	-2147483648 a …
unsigned long int	*4 bytes*	0 a 4294967295
float	*4 bytes*	10^{-38} a 10^{38}
double	*8 bytes*	10^{-308} a 10^{308}

Declaração de Variáveis

- Para armazenar um dado (valor) na memória do computador, devemos reservar o espaço correspondente ao tipo do dado;
- A declaração de uma variável reserva um espaço na memória para armazenar um dado do tipo da variável e associa o nome dessa variável a esse espaço de memória;

Declaração de Variáveis

```
int a;      // declara uma variável do tipo int
int b, c;   // declara uma variável do tipo int
float d;    // declara uma variável do tipo float

a = 5;      // armazena o valor 5 em a
b = 10;     // armazena o valor 10 em b
c = a;      // armazena o valor de a em c
d = 5.3;    // armazena o valor 5.3 em d
```



Valores Constantes

- É comum usarmos também em códigos C++ valores constantes. Por exemplo quando escrevemos a atribuição:

  a = b + 123;

- Sendo *a* e *b* variáveis previamente declaradas, deve-se representar internamente também a constante 123, para que a expressão possa ser avaliada em tempo de execução;

Valores Constantes

- *true* constante bool
- *'a'* constante char
- *123* constante int
- *12.45* constante double
- *1245e-2* constante double
- *12.45f* constante float
- *"aula de cgi"* constante char[]

Operadores

- Operadores permitem combinar variáveis e constantes, formando expressões.
- Principais tipos de operadores:
  - Aritméticos;
  - Atribuição;
  - Incremento e Decremento;
  - Relacionais e Lógicos;
  - Conversão de tipo;
  - Acesso a Memória (ponteiros e referências)
  - Alocação de Memória;

Operadores Aritméticos

- Os operadores aritméticos são:
  - Soma (+)
  - Subtração (-)
  - Multiplicação (*)
  - Divisão (/)
  - Resto (%)
  - Menos (-) *(operador unário)*

Operadores Aritméticos

```
int     a;
double  b, c;

// ...

a = 3.5;

b = a / 2.0;

c = 1 / 3 + b;
```

- As operações são feitas na precisão dos operandos. Por exemplo, a expressão 5/2 retorna o valor 2 e não 2.5, pois a expressão 5/2 opera sobre duas constantes inteiras;
- Quais os valores de a, b e c no código ao lado ?

Operadores Aritméticos

- Operador resto: x % 2
  - Expressão = 0 *(x é par)*
  - Expressão = 1 *(x é impar)*
- Precedência:

```
double  a = 3.0, b = 2.0, c = 4.0, d = 2.0;

// ...

double value1 =  a+b  * c/d;   // a+(b*c/d)
double value2 = (a+b) * c/d;

// value1 =  7.00000
// value2 = 10.00000
```

Operadores de Atribuição

- Em C++, uma atribuição é uma expressão cujo valor resultante corresponde ao valor atribuido:

```
5 + 3;        // retorna o valor 8

a = 5;        // retorna o valor 5, além de atribuir 5 à variável a

y = x = 5;    // a ordem da avaliação é da direita para a esquerda
```

- *linha 1*: a expressão retorna 8, mas não é atribuida a nenhuma variável;
- *linha 2*: a constante 5 é atribuida à variável a através do operador de atribuição =;
- *linha 3*: a expressão *x=5* atribui 5 a *x* e retorna o valor 5, que por sua vez é atribuido a *y*;

Operadores de Atribuição

- A linguagem também permite utilizar operadores de atribuição compostos. Comandos do tipo:

  i = i + 2;

  onde a variável a esquerda do operador de atribuição também aparece a direita, podem ser escritos de forma mais compacta:

  i += 2;
- De forma análoga, também é possível utilizar atribuição composta com os operadores -, *, / e %;
- Comandos do tipo

  var *op*= expr;

  são equivalentes a:

  var = var *op* (expr);

```
x *= y + 1;   // equivale a x = x * (y + 1);
              // e não a    x = x * y + 1;
```

Operadores de Incremento

- C++ apresenta ainda dois operadores que servem para incrementar e decrementar uma unidade nos valores armazenados nas variáveis:

```
n++;        // incrementa n de uma unidade
n--;        // decrementa n de uma unidade
```

- Esses operadores podem ser utilizados de forma pré-fixada ou pós-fixada. Para as expressões abaixo suponha que a variável *n* esteja em ambos os casos com valor 5:

```
x = n++;  // atribui 5 a x;
x = ++n;  // atribui 6 a x;
// em ambos os casos a variável n passa a valer 6.
```

Operadores Relacionais e Lógicos

- Operadores relacionais são usados para comparar dois valores:

<	*menor que*
>	*maior que*
< =	*menor ou igual que*
> =	*maior ou igual que*
= =	*igual a*
! =	*diferente de*

Operadores Lógicos e Relacionais

- Esses operadores comparam 2 valores. O resultado produzido por um operador relacional é 1 (*true*) ou 0 (*false*);
- Os operadores lógicos servem para combinar expressões booleanas:

&&	*operador binário E (AND)*
||	*operador binário OU (OR)*
!	*operador unário de NEGAÇÃO (NOT)*

Operadores Relacionais e Lógicos

- Expressões conectadas por && e || são avaliadas da esquerda para a direita e a avaliação se encerra assim que a veracidade ou falsidade da expressão for conhecida;
- Recomenda-se o uso de parênteses em expressões que combinam esses operadores;
- Esses operadores são normalmente utilizados em tomada de decisões (na sequencia);

```
int a, b;
int c = 23;
int d = c + 4;

a = (c < 20) || (d > c);      // verdadeiro
b = (c < 20) && (d > c);      // falso
```

Conversão de Tipo

- Em C++ existem conversões automáticas de valores na avaliação de uma expressão:

  3.0 / 2 ⟶ *3.0 / 2.0* ⟶ *1.5*

- Quando em uma atribuição o tipo do valor atribuido é diferente do tipo da variável, também há uma conversão automática (implícita) de tipo:

  float a = 3; ⟶ *float a = 3.0f;*

Conversão de Tipo

- É possível também se fazer conversões explícitas de tipo usando o operador cast:

```
int a, b;

a = (int) 3.5;                    // a recebe o valor 3.
b = (int) 3.0 / 2.0;              // b recebe o valor 1.
```

Precedência dos Operadores

Operador	Associatividade
( ) [ ] -> .	esquerda para direita
! ~ ++ -- - (*tipo*) * & sizeof(*tipo*)	direita para esquerda
* / %	esquerda para direita
+ -	esquerda para direita
<< >>	esquerda para direita
< <= > >=	esquerda para direita
== !=	esquerda para direita
&	esquerda para direita
^	esquerda para direita
|	esquerda para direita
&&	esquerda para direita
||	esquerda para direita
?:	direita para esquerda
= += -= etc.	direita para esquerda
,	esquerda para direita

Controle de Fluxo

- Tomada de decisão
- Estruturas de bloco
- Operador condicional
- Construções com laços
- Interrupções
- Seleção

Controle de Fluxo

- Até aqui apresentamos trecho de código cuja execução era sempre sequencial;
- O objetivo agora é descrever mecanismos que permitam controlar e alterar o fluxo sequencial até aqui apresentado;
- Os principais mecanismos existentes em C++ são: tomada de decição (*if-else*) e laços com teste de encerramento (*while*, *for*, *do-while*);

Tomada de Decisão

- O comando ***if*** é o comando básico para codificar tomada de decisão em C++. Sua forma pode ser:

```
if( expr )
{
  // bloco de comandos 1.
}
```

```
if( expr )
{
  // bloco de comandos 1.
}
else
{
  // bloco de comandos 2.
}
```

  - Se o resultado da avaliação resultar em um valor diferente de 0 (*false*), ou seja, se a expressão expr for verdadeira, o bloco de comandos 1 será executado;
  - A inclusão do else requisita a execução do bloco de comandos 2 se a expressão resultar em 0 (*false*);
  - Se dentro do bloco de comandos tivermos apenas 1 comando, as chaves não são necessárias:

```
if( expr )
  // comando 1;
else
  // comando 2;
```

Tipo *Boolean* e Expressões Lógicas

- Para manter compatibilidade com C Puro, que não apresenta o tipo *boolean*, C++ adota como valor de retorno de expressões lógicas o valor 0 (*false*) e 1 (*true*).
- O tipo *boolean* só pode receber 2 valores (*true* ou *false*), mas a conversão de inteiros para *booleans* em C++ é automática:

```
bool b = 7;      // (7 != 0), então b recebe true
int i = true;    // i recebe o valor 1 inteiro
```

Tomada de Decisão (*if-else*)

- O código abaixo ilustra o uso de comandos ***if*** para informar se um número digitado é par ou impar:

```
#include <iostream>

using namespace std;

int main( void )
{

  int a, b;

  cout << "Digite um número inteiro:\t";
  cin >> a;

  if(a%2 == 0)
  {
    cout << "o valor digitado é par !" << endl;
  }
  else
  {
    cout << "o valor digitado é impar !" << endl;
  }

  return 1;
}
```

Tomada de Decisão (*if-else*)

- O mesmo código pode ser utilizado sem a necessidade do uso dos parênteses na estrutura do ***if***

```
#include <iostream>

using namespace std;

int main( void )
{

  int a, b;

  cout << "Digite um número inteiro:\t";
  cin >> a;

  if(a%2 == 0)
    cout << "o valor digitado é par !" << endl;
  else
    cout << "o valor digitado é impar !" << endl;

  return 1;
}
```

Tomada de Decisão (*if-else*)

- Podemos aninhar comandos ***if***:

```
#include <iostream>

using namespace std;

int main( void )
{

  int a, b;

  cout << "Entre com dois números inteiros:\t";
  cin >> a >> b;

  if( a%2 == 0 )
  {
    if( b%2 == 0 )
    {
      cout << "os valores digitados são pares !" << endl;
    }
  }

  return 1;
}
```

Tomada de Decisão (*if-else*)

- Outra construção possível seria:

```
#include <iostream>

using namespace std;

int main( void )
{

  int a, b;

  cout << "Digite dois números inteiros:\t";
  cin >> a >> b;

  if( (a%2 == 0) && (b%2 == 0) )
  {
      cout << "os valores digitados são pares !" << endl;
  }

  return 1;
}
```

Tomada de Decisão (*else-if*)

- C++ não apresenta o comando elseif, mas ele pode ser simulado da seguinte forma:

```
#include <iostream>

using namespace std;

int main( void )
{

  cout << "Digite a temperatura: " << endl;

  int temp;
  cin >> temp;

  if( temp < 10 )
    cout << "Temperatura muito fria !" << endl;
  else if( temp < 20 )
    cout << "Temperatura fria !" << endl;
  else if( temp < 30 )
    cout << "Temperatura agradável !" << endl;
  else
    cout << "Temperatura muito quente !" << endl;

  return 1;
}
```

Estruturas de Bloco

- Cada chave aberta e fechada em C++ representa um bloco;
- Uma variável declarada dentro do bloco só pode ser acessada de dentro desse bloco;
- Em C Puro é exigida a declaração no início dos blocos, mas em C++ isso não é necessário e a variável só poderá ser acessada após a sua declaração:

```
... // i não existe nesse ponto do programa

if( n > 0 )
{
   ... // i não existe nesse ponto do programa

   int i;
   ...
}

... // i não existe nesse ponto do programa
```

- É uma boa prática de programação declarar as variáveis o mais próximo possível dos seus usos;

Construções com Laços

- Em programas computacionais procedimentos iterativos (executados em vários passos) são muito comuns;
- Um exemplo de processo iterativo é o cálculo do valor do fatorial de um número inteiro não negativo:

  n! = n x (n-1) x (n-2) x … x 3 x 2 x 1

  onde 0! = 1

Construções com Laços

- Utiliza-se nesse caso um processo iterativo onde o valor da variável varia de 1 até n, avaliando o produtório;
- A linguagem C++ apresenta oferece diversas construções possíveis para a realização de laços iterativos:
  - *while;*
  - *for;*
  - *do while;*

Laços: Comando *while*

- Forma geral:

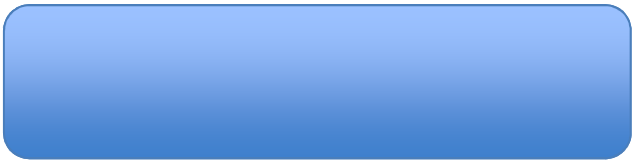
- Se o resultado da avaliação expr resultar em verdadeiro o bloco de comandos é executado.
- Ao final do bloco a expressão expr volta a ser avaliada e, enquanto expr resultar em verdadeiro o bloco de comandos é executado repetidamente.
- Quando expr for avaliada em falso o bloco de comandos deixa de ser executado e o programa segue a sua sequência natural;

Laços: Fatorial Versão *while*

```
// Fatorial versão 1

#include <iostream>

using namespace std;

int main( void )
{
  cout << "Digite um número inteiro não negativo: ";

  int n; cin >> n;

  // calcula fatorial
  int i = 1;
  int f = 1;

  while( i <= n )
  {
    f *= i;
    i++;
  }

  cout << "Fatorial = " << f << endl;

  return 0;
}
```

Laços: Comando *for*

- Uma segunda forma de construção de laços em C++ é com laços for. Sua forma geral é:



- A construção com for é equivalente ao uso do while, como segue:

Laço: Fatorial Versão *for*

```
// Fatorial versão 2

#include <iostream>

using namespace std;

int main( void )
{
  cout << "Digite um número inteiro não negativo: ";

  int n; cin >> n;

  // calcula fatorial
  int f = 1;

  for( int i=1; i <= n; i++ )
    f *= i;

  cout << "Fatorial = " << f << endl;

  return 0;
}
```

Interrupções com ***break*** e *continue*

- C++ oferece ainda duas formas para interrupção antecipada de um laço;
- O comando break, quando utilizado dentro de um laço, interrompe e encerra a sua execução:

```
#include <iostream>

using namespace std;

int main( void )
{
  int f = 1;

  for( int i=0; i < 10; i++ )
  {
    if( i == 5 )
      break;
    cout << i << "\t";
  }

  cout << "Fim" << endl;

  return 0;
}
```

- A saída desse programa quando executado será:

  0 1 2 3 4 fim

Interrupções com *break* e ***continue***

- O comando continue também interrompe a execução dos comandos de um laço, porém, nesse caso, o laço não é automaticamente finalizado;
- O comando continue interrompe a execução de um laço para a próxima iteração:

```
#include <iostream>

using namespace std;

int main( void )
{
  int f = 1;

  for( int i=0; i < 10; i++ )
  {
    if( i == 5 )
      continue;
    cout << i << "\t";
  }

  cout << "Fim" << endl;

  return 0;
}
```

- A saída desse programa quando executado será:

  0 1 2 3 4 6 7 8 9 fim

Seleção com ***switch***

- C++ prove o comando *switch* para selecionar um entre um conjunto de casos possíveis. Sua forma geral é:

```
switch( expr )
{
  case op1:
    ...  // bloco de comandos se expr == op1
    break;
  case op2:
    ...  // bloco de comandos se expr == op2
    break;
  case op3:
    ...  // bloco de comandos se expr == op3
    break;
  default:
    ...  // executados se expr for diferente de todos
    break;
}
```

Seleção com ***switch***

- op_i deve ser um número inteiro ou uma constante caractere. Se *expr* resultar no valor op_i os comandos seguites ao case op_i serão executados até encontrar um *break*;
- Se o comando break for omitido, a execução do *case* continua com a execução do *case* seguinte;
- Se o valor de *expr* for diferente de todos os *cases*, o bloco de comandos associado a *default* (que pode ser omitido) é executado ;

Seleção com ***switch***

```
// Calculadora de quatro operações

#include <iostream>

using namespace std;

int main( void )
{
  char  op;
  float num1, num2;

  cout << "Digite: numero op numero" << endl;
  cin  >> num1 >> op >> num2;

  switch( op )
  {
    case '+':
      cout << num1 + num2 << endl;
      break;
    case '-':
      cout << num1 - num2 << endl;
      break;
    case '*':
      cout << num1 * num2 << endl;
      break;
    case '/':
      cout << num1 / num2 << endl;
      break;
    default:
      cout << "Operador invalido !" << endl;
      break;
  }

  return 0;
}
```

Pesquisa Binária

```
#include <iostream>

using namespace std;

int main( void )
{

 cout << "Tamanho do vetor (maximo 10):\t";
 int n;
 cin >> n;
 if( n > 10 )
  n = 10;
 cout << endl;

 cout << "Vetor de inteiros ordenados crescente:" << endl;
 int v[10];
 for( int i = 0; i < n; i++ )
 {
   cout << "Valor " << "v[" << i << "] =\t";
   cin >> v[i];
 }
 cout << endl;

 cout << "Numero inteiro a ser procurado:\t";
 int x;
 cin >> x;
```

```
 int low = 0;
 int high = n - 1;
 while( low <= high )
 {
  int mid;
  mid = (low + high) / 2;
  if( x < v[mid] )
  {
   high = mid - 1;
  }
  else if( x > v[mid] )
  {
   low = mid + 1;
  }
  else
  {
   cout << "Indice do numero no vetor:\t" << mid << endl;
   return( 0 );
  }
 }

 cout << "Numero não encontrado!" << endl;

 return( 0 );
}
```