C++
Básico ao Avançado
Tipo assim...

Heitor Rodrigues Savegnago

UFABC Rocket Design

2017.3

Tipos

Tipos

- Todo programa precisa armazenar dados, até o código mínimo utiliza uma

Tipos

- Todo programa precisa armazenar dados, até o código mínimo utiliza uma
- quantidade de memória.

Tipos

- Todo programa precisa armazenar dados, até o código mínimo utiliza uma
- quantidade de memória.
- Podem ser armazenados dados numéricos, de texto, estados lógicos.

Tipos

- Todo programa precisa armazenar dados, até o código mínimo utiliza uma
- quantidade de memória.
- Podem ser armazenados dados numéricos, de texto, estados lógicos.
- Os tipos aqui apresentados são denominados *tipos primitivos*.

Tipos

Tipos

- Estados lógicos são armazenados nas memórias tipo `bool`

Tipos

- Estados lógicos são armazenados nas memórias tipo `bool`
- Caracteres de texto são armazenados nas memórias tipo `char`

Tipos

- Estados lógicos são armazenados nas memórias tipo `bool`
- Caracteres de texto são armazenados nas memórias tipo `char`
- Valore numéricos apresentam dois tipos de armazenamento:

Tipos

- Estados lógicos são armazenados nas memórias tipo `bool`
- Caracteres de texto são armazenados nas memórias tipo `char`
- Valore numéricos apresentam dois tipos de armazenamento:
  - Números inteiros nos tipo `int`

Tipos

- Estados lógicos são armazenados nas memórias tipo `bool`
- Caracteres de texto são armazenados nas memórias tipo `char`
- Valore numéricos apresentam dois tipos de armazenamento:
  - Números inteiros nos tipo `int`
  - Números flutuantes nos tipo `float` e `double`

Tipos

- Estados lógicos são armazenados nas memórias tipo `bool`
- Caracteres de texto são armazenados nas memórias tipo `char`
- Valore numéricos apresentam dois tipos de armazenamento:
  - Números inteiros nos tipo `int`
  - Números flutuantes nos tipo `float` e `double`

Há ainda um tipo especial, que não armazena dados, o tipo `void`, seu será explicado daqui algumas aulas.

Modificadores

Modificadores

- Os modificadores de faixa são palavras-chave que alteram os valores registráveis por um tipo.

Modificadores

- Os modificadores de faixa são palavras-chave que alteram os valores registráveis por um tipo.
- Os modificadores `signed` e `unsigned` alteram a signatura da faixa.

Modificadores

- Os modificadores de faixa são palavras-chave que alteram os valores registráveis por um tipo.
- Os modificadores `signed` e `unsigned` alteram a signatura da faixa.
- Os modificadores `short` e `long` alteram o comprimento da faixa.

Modificadores

Tabela: Relação de faixa e tamanhos de memória para tipos primitivos com modificadores de faixa

código	tamanho (B)	valor mínimo	valor máximo

Modificadores

Tabela: Relação de faixa e tamanhos de memória para tipos primitivos com modificadores de faixa

código	tamanho (B)	valor mínimo	valor máximo
`bool`	1	0	1

Modificadores

Tabela: Relação de faixa e tamanhos de memória para tipos primitivos com modificadores de faixa

código	tamanho (B)	valor mínimo	valor máximo
`bool`	1	0	1
`signed char`	1	-127	126
`char`	1	-127	126
`unsigned char`	1	0	255

Modificadores

Tabela: Relação de faixa e tamanhos de memória para tipos primitivos com modificadores de faixa

código	tamanho (B)	valor mínimo	valor máximo
`bool`	1	0	1
`signed char`	1	-127	126
`char`	1	-127	126
`unsigned char`	1	0	255
`signed short int`	2	-32768	32767
`short int`	2	-32768	32767
`unsigned short int`	2	0	64535

Modificadores

Tabela: Relação de faixa e tamanhos de memória para tipos primitivos com modificadores de faixa

código	tamanho (B)	valor mínimo	valor máximo
`bool`	1	0	1
`signed char`	1	-127	126
`char`	1	-127	126
`unsigned char`	1	0	255
`signed short int`	2	-32768	32767
`short int`	2	-32768	32767
`unsigned short int`	2	0	64535
`signed int`	4	-2147483648	2147483647
`int`	4	-2147483648	2147483647
`unsigned int`	4	0	4294967295

Modificadores

Tabela: Relação de faixa e tamanhos de memória para tipos primitivos com modificadores de faixa

código	tamanho (B)	valor mínimo	valor máximo
`bool`	1	0	1
`signed char`	1	-127	126
`char`	1	-127	126
`unsigned char`	1	0	255
`signed short int`	2	-32768	32767
`short int`	2	-32768	32767
`unsigned short int`	2	0	64535
`signed int`	4	-2147483648	2147483647
`int`	4	-2147483648	2147483647
`unsigned int`	4	0	4294967295
`signed long int`	8	-9223372036854775808	9223372036854775807
`long int`	8	-9223372036854775808	9223372036854775807
`unsigned long int`	8	0	18446744073709551616

Modificadores

Tabela: Relação de faixa e tamanhos de memória para tipos primitivos com modificadores de faixa

código	tamanho (B)	valor mínimo	valor máximo
`bool`	1	0	1
`signed char`	1	-127	126
`char`	1	-127	126
`unsigned char`	1	0	255
`signed short int`	2	-32768	32767
`short int`	2	-32768	32767
`unsigned short int`	2	0	64535
`signed int`	4	-2147483648	2147483647
`int`	4	-2147483648	2147483647
`unsigned int`	4	0	4294967295
`signed long int`	8	-9223372036854775808	9223372036854775807
`long int`	8	-9223372036854775808	9223372036854775807
`unsigned long int`	8	0	18446744073709551616
`float`	4	$1.2 \cdot 10^{-38}$	$3.4 \cdot 10^{+38}$

Modificadores

Tabela: Relação de faixa e tamanhos de memória para tipos primitivos com modificadores de faixa

código	tamanho (B)	valor mínimo	valor máximo
`bool`	1	0	1
`signed char`	1	-127	126
`char`	1	-127	126
`unsigned char`	1	0	255
`signed short int`	2	-32768	32767
`short int`	2	-32768	32767
`unsigned short int`	2	0	64535
`signed int`	4	-2147483648	2147483647
`int`	4	-2147483648	2147483647
`unsigned int`	4	0	4294967295
`signed long int`	8	-9223372036854775808	9223372036854775807
`long int`	8	-9223372036854775808	9223372036854775807
`unsigned long int`	8	0	18446744073709551616
`float`	4	$1.2 \cdot 10^{-38}$	$3.4 \cdot 10^{+38}$
`double`	8	$1.73 \cdot 10^{-308}$	$1.7 \cdot 10^{+308}$
`long double`	16	$3.4 \cdot 10^{-4932}$	$3.4 \cdot 10^{+4932}$

Variáveis

De nada adianta *existir* um tipo de armazenamento de dados se não soubermos usá-lo.

Variáveis

De nada adianta *existir* um tipo de armazenamento de dados se não soubermos usá-lo.
Uma variável é declarada colocando a lista de modificadores, o tipo e o nome da variável.

Variáveis

De nada adianta *existir* um tipo de armazenamento de dados se não soubermos usá-lo.
Uma variável é declarada colocando a lista de modificadores, o tipo e o nome da variável.

```
<modificadores> <tipo> <nome>;
```

Variáveis

De nada adianta *existir* um tipo de armazenamento de dados se não soubermos usá-lo.
Uma variável é declarada colocando a lista de modificadores, o tipo e o nome da variável.

```
<modificadores> <tipo> <nome>;
```

Quando uma variável é declarada, ela pode vir com lixo de memória, para evitar isso, declaramos a variável com um valor de inicialização, seguindo a sintaxe:

Variáveis

De nada adianta *existir* um tipo de armazenamento de dados se não soubermos usá-lo.
Uma variável é declarada colocando a lista de modificadores, o tipo e o nome da variável.

```
<modificadores> <tipo> <nome>;
```

Quando uma variável é declarada, ela pode vir com lixo de memória, para evitar isso, declaramos a variável com um valor de inicialização, seguindo a sintaxe:

```
<modificadores> <tipo> <nome>(<valor>);
```

Variáveis

```
1  bool falso(0);              //Com número
2  bool verdadeiro(true);      //Com palavra-chave
3  char igual(0x3D);           //Sinal de igual ASCII
4  char letraA('A');           //Aspas simples
5
6  int contador(1), acumulador(0);    //Várias variáveis do mesmo
       tipo
7  unsigned int positivo(523);        //Inteiro sem sinal
8  short doisBytes(93);               //Modificador de
       comprimento
9  long grande(32416189349L);         //Número grande
10 double cargaFundamental(1.6e-19); //Notação científica
11 float pi(3.14159265358979323846264338327950288419f);  //
       Flutuante preciso
```

Variáveis

```
1  bool falso(0);              //Com número
2  bool verdadeiro(true);      //Com palavra-chave
3  char igual(0x3D);           //Sinal de igual ASCII
4  char letraA('A');           //Aspas simples
5
6  int contador(1), acumulador(0);    //Várias variáveis do mesmo
       tipo
7  unsigned int positivo(523);        //Inteiro sem sinal
8  short doisBytes(93);               //Modificador de
       comprimento
9  long grande(32416189349L);         //Número grande
10 double cargaFundamental(1.6e-19); //Notação científica
11 float pi(3.14159265358979323846264338327950288419f);  //
       Flutuante preciso
```

Note os detalhes!

printf

printf

- O `prinft` é uma das opções para exibição na tela.

printf

- O `prinft` é uma das opções para exibição na tela.
- Pular linha? Caracter especial!
  <en.cppreference.com/w/cpp/language/escape>

printf

- O `prinft` é uma das opções para exibição na tela.
- Pular linha? Caracter especial!
  <en.cppreference.com/w/cpp/language/escape>
- Exibir variáveis? Sequência especial!
  <en.cppreference.com/w/cpp/io/c/fprintf>

printf

- O `prinft` é uma das opções para exibição na tela.
- Pular linha? Caracter especial!
  <en.cppreference.com/w/cpp/language/escape>
- Exibir variáveis? Sequência especial!
  <en.cppreference.com/w/cpp/io/c/fprintf>

```
printf(<string>, <...>);
```

printf

O código:

```
printf("cinco + sete = %i\n", 12);
printf("%i + %i = %s\n", 5, 7, "doze");
float pi(3.1415);
printf("Pi vale %f\n", pi);
```

printf

O código:

```
printf("cinco + sete = %i\n", 12);
printf("%i + %i = %s\n", 5, 7, "doze");
float pi(3.1415);
printf("Pi vale %f\n", pi);
```

Gera a saída:

```
cinco + sete = 12
5 + 7 = doze
Pi vale 3.141500
```

Modificando variáveis

Modificando variáveis

- Variáveis que não variam não são variáveis

Modificando variáveis

- Variáveis que não variam não são variáveis
- As alterações nas variáveis dependem de seu tipo

Modificando variáveis

- Variáveis que não variam não são variáveis
- As alterações nas variáveis dependem de seu tipo
- Existem três grupos de operadores: unários, binários e ternários:

Modificando variáveis

- Variáveis que não variam não são variáveis
- As alterações nas variáveis dependem de seu tipo
- Existem três grupos de operadores: unários, binários e ternários:

Unário utiliza apenas um valor

Modificando variáveis

- Variáveis que não variam não são variáveis
- As alterações nas variáveis dependem de seu tipo
- Existem três grupos de operadores: unários, binários e ternários:

Unário utiliza apenas um valor
Binário utiliza dois valores

Modificando variáveis

- Variáveis que não variam não são variáveis
- As alterações nas variáveis dependem de seu tipo
- Existem três grupos de operadores: unários, binários e ternários:

Unário utiliza apenas um valor
Binário utiliza dois valores
Ternário utiliza três valores

Modificando variáveis

- Variáveis que não variam não são variáveis
- As alterações nas variáveis dependem de seu tipo
- Existem três grupos de operadores: unários, binários e ternários:

Unário utiliza apenas um valor
Binário utiliza dois valores
Ternário utiliza três valores

- Todo operador retorna o valor de sua operação

Sinalizadores

Sinalizadores

- Equivalem aos sinais matemáticos

Sinalizadores

- Equivalem aos sinais matemáticos
- Fazem o mesmo que multiplicar um número por $+1$ ou -1

Sinalizadores

- Equivalem aos sinais matemáticos
- Fazem o mesmo que multiplicar um número por $+1$ ou -1
- Não alteram o valor registrado

Sinalizadores

- Equivalem aos sinais matemáticos
- Fazem o mesmo que multiplicar um número por $+1$ ou -1
- Não alteram o valor registrado

```
-<name>;  //Só aparece em um contexto onde o retorno é
    utilizado

+<name>;
```

Sinalizadores

```
1 //...
2 int A(-5);  //A vale -5
3 int B(+A);  //A vale -5, B vale -5
4 int C(-A);  //A vale -5, B vale -5, C vale 5
5
6 int D(5);   //D vale 5,
7 int E(-D);  //D vale 5, E vale -5
8 int F(+D);  //D vale 5, E vale -5, F vale 5
9 //...
```

Atribuidor simples

Atribuidor simples

- O operador mais utilizado é o atribuidor simples

Atribuidor simples

- O operador mais utilizado é o atribuidor simples
- Ele atribui o valor a direita à variável a direita

Atribuidor simples

- O operador mais utilizado é o atribuidor simples
- Ele atribui o valor a direita à variável a direita
- Cuidado para não inverter

Atribuidor simples

- O operador mais utilizado é o atribuidor simples
- Ele atribui o valor a direita à variável a direita
- Cuidado para não inverter
- Não atribuir tipos a variáveis de outros tipos

Atribuidor simples

- O operador mais utilizado é o atribuidor simples
- Ele atribui o valor a direita à variável a direita
- Cuidado para não inverter
- Não atribuir tipos a variáveis de outros tipos

```
<alvo> = <item>;
```

Atribuidor simples

```
1  //...
2  int A;
3  A = 10;                //A passa a valer 10
4  float B = 5.1;         //Operadores podem ser usados na declaraç
       ão.
5
6  float C(B = 13.25); //Todo operador retorna o valor de sua
       operação
7
8  int D = A = 20;
9  /*
10   Da direita para a esquerda para a direita, A passa a valer
         10
11   Então D passa a ter o valor da operação à direita, 10
12 */
13 //...
```

Aritméticos

Aritméticos

- As quatro operações básicas da matemática

Aritméticos

- As quatro operações básicas da matemática
- Os símbolos padrão

Aritméticos

- As quatro operações básicas da matemática
- Os símbolos padrão
- Não alteram o valor registrado

Aritméticos

- As quatro operações básicas da matemática
- Os símbolos padrão
- Não alteram o valor registrado
- Cuidado para não dividir por 0

Aritméticos

- As quatro operações básicas da matemática
- Os símbolos padrão
- Não alteram o valor registrado
- Cuidado para não dividir por 0

```
<valor1> + <valor2>;  //O uso só é coerente em casos onde o
    retorno é utilizado
<valor1> - <valor2>;
<valor1> * <valor2>;
<valor1> / <valor2>;
<valor1> % <valor2>;
```

Aritméticos

- As quatro operações básicas da matemática
- Os símbolos padrão
- Não alteram o valor registrado
- Cuidado para não dividir por 0

```
<valor1> + <valor2>;  //O uso só é coerente em casos onde o
   retorno é utilizado
<valor1> - <valor2>;
<valor1> * <valor2>;
<valor1> / <valor2>;
<valor1> % <valor2>;
```

$$\begin{array}{c|c} E & F \\ \hline G & H \end{array} \Rightarrow \begin{array}{c|c} 13 & 5 \\ \hline 3 & 2 \end{array}$$

Aritméticos

```
1  //...
2  int A(45 + 5);  //A vale 50
3  int B(A - 15);  //B vale 35
4  int C(B - A);   //C vale  - 15
5
6  int D(A + B - C); //D vale o mesmo que 50 + 35 - ( - 15), ou
       seja 100
7
8  int E(13),  F(5);            //E vale 13, F vale 5
9  int G(E % F), H(E / F);      //G vale 3, H vale 2
10 int I(F * H);                //I vale 10
11 int J(I + G);                //J vale 13
12
13 float K(13.0f), L(5.0f);
14 float M(K / L);              //K vale 2.6
15 float N(L * M);              //N vale 13
16 //...
```

Comparadores

Comparadores

- Verificar se valores são iguais ou diferentes

Comparadores

- Verificar se valores são iguais ou diferentes
- Descobrir se um valor é maior que outro

Comparadores

- Verificar se valores são iguais ou diferentes
- Descobrir se um valor é maior que outro

```
<valor1> == <valor2>; //Só é coerente quando o retorno é
    utilizado
<valor1> != <valor2>;
<valor1> < <valor2>;
<valor1> > <valor2>;
<valor1> <= <valor2>;
<valor1> >= <valor2>;
```

Comparadores

```
1  //...
2  bool A(true);
3  int B(10), C(10), D(15);
4
5  bool E(B>C);     //E vale 0
6  bool F(A==E);    //F vale 0
7
8  bool G(D>=B);    //G vale 1
9  bool H(E!=G);    //H vale 1
10 bool I(B==C);    //I vale 1
11 //...
```

Vamos testar!