C++
Básico ao Avançado
Funcional

Heitor Rodrigues Savegnago

UFABC Rocket Design

2017.3

Escrita - *O*

Escrita - *O*

- Já sabemos exibir valores na tela

Escrita - *O*

- Já sabemos exibir valores na tela
- Tem uma forma muito mais fácil...

Escrita - *O*

- Já sabemos exibir valores na tela
- Tem uma forma muito mais fácil...
- Você não precisa nem saber o que vai mostrar

Escrita - *O*

- Já sabemos exibir valores na tela
- Tem uma forma muito mais fácil...
- Você não precisa nem saber o que vai mostrar
- Mas...

Escrita - *O*

- Já sabemos exibir valores na tela
- Tem uma forma muito mais fácil...
- Você não precisa nem saber o que vai mostrar
- Mas... Não é bom pra formatar a saída

Escrita - *O*

- Já sabemos exibir valores na tela
- Tem uma forma muito mais fácil...
- Você não precisa nem saber o que vai mostrar
- Mas... Não é bom pra formatar a saída
- `cout`

Escrita - *O*

- Já sabemos exibir valores na tela
- Tem uma forma muito mais fácil...
- Você não precisa nem saber o que vai mostrar
- Mas... Não é bom pra formatar a saída
- cout

```
1 #include <iostream>
2 int main()
3 {
4   std::cout << "Hellow World!" << std::endl;
5   return 0;
6 }
```

Leitura - *I*

Leitura - *I*

- E existem alguma coisa que faça leitura tão fácil assim?

Leitura - *I*

- E existem alguma coisa que faça leitura tão fácil assim?
- É amigo do programador

Leitura - *I*

- E existem alguma coisa que faça leitura tão fácil assim?
- É amigo do programador
- Você só precisa passar a variável

Leitura - *I*

- E existem alguma coisa que faça leitura tão fácil assim?
- É amigo do programador
- Você só precisa passar a variável
- Mas...

Leitura - *I*

- E existem alguma coisa que faça leitura tão fácil assim?
- É amigo do programador
- Você só precisa passar a variável
- Mas. . . A entrada depende do tipo da variável passada

Leitura - *I*

- E existem alguma coisa que faça leitura tão fácil assim?
- É amigo do programador
- Você só precisa passar a variável
- Mas. . . A entrada depende do tipo da variável passada
- `cin`

Leitura - *I*

- E existem alguma coisa que faça leitura tão fácil assim?
- É amigo do programador
- Você só precisa passar a variável
- Mas. . . A entrada depende do tipo da variável passada
- `cin`

```
1 #include <iostream>
2 int main()
3 {
4   int A;
5   std::cin >> A;
6   //...
7   return 0;
8 }
```

Por quê?

Por quê?

- Por que você me fez sofrer até agora?

Por quê?

- Por que você me fez sofrer até agora?
- Não é porque eu queira vez sua angústia

Por quê?

- Por que você me fez sofrer até agora?
- Não é porque eu queira vez sua angústia
- É necessário conhecer a base

Por quê?

- Por que você me fez sofrer até agora?
- Não é porque eu queira vez sua angústia
- É necessário conhecer a base
- `iostream` é avançado e trabalha com conceitos que não vimos

Procedural

Procedural

- Lembra do tipo `void` que você esnobava quando criança?

Procedural

- Lembra do tipo `void` que você esnobava quando criança?
- Ele cresceu...

Procedural

- Lembra do tipo `void` que você esnobava quando criança?
- Ele cresceu...
- Procedimentos são blocos de comandos, nomeados

Procedural

- Lembra do tipo `void` que você esnobava quando criança?
- Ele cresceu...
- Procedimentos são blocos de comandos, nomeados
- Podem ser invocados em qualquer lugar do programa

Procedural

- Lembra do tipo `void` que você esnobava quando criança?
- Ele cresceu...
- Procedimentos são blocos de comandos, nomeados
- Podem ser invocados em qualquer lugar do programa
- Melhor que copiar sequências

Procedural

- Lembra do tipo `void` que você esnobava quando criança?
- Ele cresceu...
- Procedimentos são blocos de comandos, nomeados
- Podem ser invocados em qualquer lugar do programa
- Melhor que copiar sequências
- Deixa o código mais legível

Procedural

- Lembra do tipo `void` que você esnobava quando criança?
- Ele cresceu...
- Procedimentos são blocos de comandos, nomeados
- Podem ser invocados em qualquer lugar do programa
- Melhor que copiar sequências
- Deixa o código mais legível
- Facilita na hora de achar erros

Procedural

- Lembra do tipo `void` que você esnobava quando criança?
- Ele cresceu...
- Procedimentos são blocos de comandos, nomeados
- Podem ser invocados em qualquer lugar do programa
- Melhor que copiar sequências
- Deixa o código mais legível
- Facilita na hora de achar erros
- Podem trabalhar com variáveis globais

Procedural

- Lembra do tipo `void` que você esnobava quando criança?
- Ele cresceu...
- Procedimentos são blocos de comandos, nomeados
- Podem ser invocados em qualquer lugar do programa
- Melhor que copiar sequências
- Deixa o código mais legível
- Facilita na hora de achar erros
- Podem trabalhar com variáveis globais
- Exige um bloco...

Procedural

- Lembra do tipo `void` que você esnobava quando criança?
- Ele cresceu...
- Procedimentos são blocos de comandos, nomeados
- Podem ser invocados em qualquer lugar do programa
- Melhor que copiar sequências
- Deixa o código mais legível
- Facilita na hora de achar erros
- Podem trabalhar com variáveis globais
- Exige um bloco...

```
1 void foo(void)
2 {
3   <comand1>;
4   //...
5   <comandN>;
6 }
```

Euler

Euler

```
1  int A, B;
2
3  void MDCab(void)
4  {
5    while(B!=0)
6    {
7      int R(A%B);
8      A=B;
9      B=R;
10   }
11 }
12
13 int main()
14 {
15   A = 93;
16   B = 45;
17   MDCab();
18   int C(A); //C vale 3
19   return 0;
20 }
```

Paramétrico

Paramétrico

- Nem sempre ler variável global é útil

Paramétrico

- Nem sempre ler variável global é útil
- Nem sempre existir variável gobal é útil

Paramétrico

- Nem sempre ler variável global é útil
- Nem sempre existir variável gobal é útil
- Parametros entram como argumentos na forma de variáveis

Paramétrico

- Nem sempre ler variável global é útil
- Nem sempre existir variável gobal é útil
- Parametros entram como argumentos na forma de variáveis
- Repeite a ordem

Paramétrico

- Nem sempre ler variável global é útil
- Nem sempre existir variável gobal é útil
- Parametros entram como argumentos na forma de variáveis
- Repeite a ordem
- Separação por vírgula

Paramétrico

- Nem sempre ler variável global é útil
- Nem sempre existir variável gobal é útil
- Parametros entram como argumentos na forma de variáveis
- Repeite a ordem
- Separação por vírgula
- Ainda devolve o valor por variável global

Paramétrico

- Nem sempre ler variável global é útil
- Nem sempre existir variável gobal é útil
- Parametros entram como argumentos na forma de variáveis
- Repeite a ordem
- Separação por vírgula
- Ainda devolve o valor por variável global

```
1 void foo(<tipo1> <var1>, ... , <tipoN> <varN>)
2 {
3   <comand1>;
4   //...
5   <comandM>;
6 }
```

Euler

Euler

```
1  int C(0);
2
3  void MDCab(int A, int B)
4  {
5    while(B!=0)
6    {
7      int R(A%B);
8      A=B;
9      B=R;
10   }
11   C = A;
12 }
13
14 int main()
15 {
16   int A(93);
17   int B(45);
18   MDCab(A,B);
19   //C vale 3
20   return 0;
21 }
```

Funcional

Funcional

- Devolver valor calculado

Funcional

- Devolver valor calculado
- Pode ser invocadas em qualquer lugar do programa

Funcional

- Devolver valor calculado
- Pode ser invocadas em qualquer lugar do programa
- Leva o tipo do retorno

Funcional

- Devolver valor calculado
- Pode ser invocadas em qualquer lugar do programa
- Leva o tipo do retorno
- Palavra-chave:

Funcional

- Devolver valor calculado
- Pode ser invocadas em qualquer lugar do programa
- Leva o tipo do retorno
- Palavra-chave: o `return`

Funcional

- Devolver valor calculado
- Pode ser invocadas em qualquer lugar do programa
- Leva o tipo do retorno
- Palavra-chave: o `return`
- Retorno é obrigatório

Funcional

- Devolver valor calculado
- Pode ser invocadas em qualquer lugar do programa
- Leva o tipo do retorno
- Palavra-chave: o `return`
- Retorno é obrigatório
- Porque o `main` tem tipo `int` e retorna `0`

Funcional

- Devolver valor calculado
- Pode ser invocadas em qualquer lugar do programa
- Leva o tipo do retorno
- Palavra-chave: o `return`
- Retorno é obrigatório
- Porque o `main` tem tipo `int` e retorna 0

```
<tipo> foo(<tipo1> <var1>, ... , <tipoN> <varN>)
{
  <comand1>;
  //...
  <comandM>;
  return <retorno>;
}
```

Euler

Euler

```
1  void MDCab(int A, int B)
2  {
3    while(B!=0)
4    {
5      int R(A%B);
6      A=B;
7      B=R;
8    }
9    return B;
10 }
11
12 int main()
13 {
14   int A(93);
15   int B(45);
16   int C(MDCab(A,B));
17   return 0;
18 }
```

Vamos testar!