Resumo

Marco A L Barbosa malbarbo.pro.br

1 Compilação

Criar executável `ola` a partir do arquivo `ola.cpp`

```
g++ -o ola ola.cpp
```

2 Linguagem

2.1 Notação

`[]`: significa que a construção que está entre os colchetes é opcional e portando não precisa ser especificada.

`...`: significa que a construção que veio antes pode ser repetida.

2.2 Variáveis

Forma geral

```
Tipo nome_variavel [= valor inicial];
```

Forma de execução

O computador aloca uma célula de memória e associa o nome `nome_variavel` com a célula. Se a expressão "valor inicial" existir, ela é avaliada e o valor resultante é armazenado na célula `nome_variavel`. Note que se um valor inicial não for fornecido, o valor armazenado na célula de memória é indefinido.

Exemplos

```
int a = 10;
// aloca uma célula de memória e associa com o nome b e armazena o valor 22 nessa célula
int b = a * a + 2;
double x;
```

Note que as instruções são executadas sequencialmente, uma após a outra.

2.3 Funções

Forma geral

```
TipoSaida nome_da_funcao(TipoEntrada1 e1, ...)
{
    [instruções; ...]
    return resultado;
}
```

Forma de execução

Para executar uma função, o fluxo de execução é desviado para o início da função e as instruções da função são executadas em ordem. No final, quando a instrução `return` é executada, o fluxo de execução volta para o local onde a função foi chamada.

Exemplos

```cpp
// inverte_unidade_dezena(4367) -> 4376
bool inverte_unidade_dezena(int num)
{
    int u = num % 10;
    int d = (num / 10) % 10;
    int restante = num / 100;
    return restante * 100 + u * 10 + d;
}

int main()
{
    int x = inverte_unidade_dezena(4367); // 4376
    cout << x << endl;
}
```

Nesse exemplo as linhas são executadas na ordem 12, 4, 5, 6, 7, 12, 13.

2.4 Instrução `if/else`

Forma geral

```
if (condição) {
    instruções então;
} [else {
    instruções senão;
}]
```

Forma de execução

O computador avalia a "condição" e verifica o resultado. Se o resultado for **`true`**, então as instruções do bloco "instruções então" são executadas, senão (o resultado é **`false`**), as instruções do bloco "instruções senão" são executadas (se elas foram especificadas).

Exemplos

```cpp
int main()
{
    int a = 10;
    int b = 20;
    int m;
    if (a > b) {
        m = a;
    } else {
        m = b;
    }
    cout << m << endl;
}
```

O valor exibido por esse programa é 20 e as linhas são executadas na ordem 3, 4, 5, 6, 9, 11.

2.5 Tipos enumerados

Forma geral

```
enum NomeEnum {
    ValorEnum,
    ...
};
```

Quando utilizar?

Quando todos os valores válidos para o tipo podem ser nomeados.

Exemplos

A cor de um semáforo só pode assumir 3 valores: vermelho, amarelo ou verde, então criamos um tipo enumerado.

```
// Representa a cor de um semáforo
enum Cor {
  Vermelho,
  Amarelo,
  Verde
};
```

2.6 Instrução `switch/case`

Forma geral

```
switch (expressão) {
    [case caso:
        instruções;
        break;
    ]...
    [default:
        instruções;
        break;]
}
```

Forma de execução

A "expressão" é avaliada e seu valor é comparado com cada caso na sequência.

Quando um "caso" que tem o mesmo valor do resultado da expressão é encontrado, as "instruções" daquele caso são executadas até encontrar um `break`, então a instrução **`switch/case`** termina e o programa continua a execução com a próxima instrução após o **`switch/case`**.

Se o valor da expressão não é igual a nenhum "caso", então as instruções da cláusula **`default`** são executadas (se elas foram especificadas).

Exemplo

```
1  int main()
2  {
3      Cor atual = Vermelho;
4      Cor proxima;
5      switch (c) {
6          case Verde:
7              proxima = Amarelo;
8              break;
9          case Vermelho:
10             proxima = Verde;
11             break;
12         case Amarelo:
13             proxima = Vermelho;
14             break;
15     }
16     cout << repr(proxima) << endl;
17 }
```

O valor exibido por esse programa é Vermelho e as linhas são executadas na ordem 3, 4, 5, 6, 9, 10, 11, 16.

2.7 Tipos compostos

Forma geral

```
struct NomeStruct {
    Tipo campo_struct;
    ...
};
```

Quando utilizar?

Quando a informação consiste de dois ou mais itens que juntos descrevem uma entidade.

Exemplos

```
// Representado o resultado de um Jogo.
struct ResultadoJogo {
    string time_anfitriao;
    int gols_anfitriao;
    string time_visitante;
    int gols_visitante;
};

int main()
{
    ResultadoJogo r = { "Brazil", 1, "Argentina", 0 };
    cout << "O resultado do jogo foi:" << endl;
    cout << r.time_anfitriao << " " << r.gols_anfitriao << endl;
    cout << r.time_visitante << " " << r.gols_visitante << endl;
}
```

Este programa produz a seguinte saída:

```
O resultado do jogo foi:
Brazil 1
Argentina 0
```

2.8 Instrução `for each`

Forma geral

```
for (Tipo nome_variavel : arranjo) {
    instruções;
}
```

Forma de execução

- O primeiro valor de `arranjo` é atribuído para `nome_varialve` e as `instruções` são executadas;
- O segundo valor de `arranjo` é atribuído para `nome_variavel` e as `instruções` são executadas;
- ...
- E assim por diante até que todos os valores de `arranjo` tenham sidos atribuídos para `nome_variavel`.

Exemplo

```
1 int main()
2 {
3     vector<int> valores = {3, 1, 5, 2};
4     int soma = 0;
5     for (int v : valores) {
6         soma = soma + v;
7     }
```

```
    cout << soma << endl;
}
```

Este programa exibe o valor 11 e as linhas são executadas na ordem 3, 4, 5, 6, 5, 6, 5, 6, 5, 6, 5, 8.

2.9 Instrução `for`

Forma geral

```
for (inicialização; condição; atualização) {
   instruções;
}
```

Forma de execução

- A "inicialização" é executada;
- Em seguida a "condição" é verificada, se ela for `true` as "instruções" são executadas, senão a repetição termina;
- Depois a "atualização" é executada e o passo anterior é executado novamente.

Exemplo

```
int main()
{
    vector<int> valores = {1, 3, 2, 4, 5};
    bool em_ordem = true;
    for (int i = 1; i < valores.size() && em_ordem; i = i + 1) {
        if (valores[i - 1] > valores[i]) {
            em_ordem = false
        }
    }
    cout << em_ordem << endl;
}
```

Este programa exibe o valor 0 (false) e as linhas são executadas na ordem 3, 4, 5, 6, 5, 6, 7, 5, 10.

2.10 Instrução `while`

Forma geral

```
while (condição) {
    instruções;
}
```

Forma de execução

A condição é avaliada, se o resultado for `true`, as "instruções" são executadas e o processo começa novamente, senão, a repetição é finalizada.

Exemplo

```
int main()
{
    int num = 9;
    int primo = true;
    int divisor = 2;
    while (divisor < num / 2 && primo) {
        if (num % divisor == 0) {
            primo = false;
        }
        divisor = divistor + 1;
    }
```

```
12      cout << primo << endl;
13  }
```

Este programa exibe o valor 0 (false) e as linhas são executadas na ordem 3, 4, 5, 6, 7, 10, 6, 7, 8, 10, 6, 12.

3 Tipos primitivos e pré-definidos

3.1 Números

`int`: números inteiros no intervalo -2^{32} a $2^{32} - 1$.

`double`: aproximação de número decimais com até 15 casas.

Operações que misturam `double` e `int` produzem `double` como resposta.

```
// Operações com int
23 + 12; // 35
23 - 12; // 11
3 * 6 ; // 18
// Divisão inteira
15 / 5; // 3
15 / 7; // 2
// Módulo
15 % 5; // 0
15 % 7; // 1
18 % 7; // 4

// Operações com double
1.2 + 2.5; // 3.7
4.2 - 8.0; // -3.8
4.1 * 2.0; // 8.2
4.75 / 0.5; // 9.5
```

Função da biblioteca `cmath`.

```
// Produz o maior inteiro menor ou igual a n (piso).
double floor(double n);
// Exemplos
floor(1.0); // 1.0
floor(1.3); // 1.0
floor(1.5); // 1.0
floor(1.7); // 1.0

// Produz o menor inteiro menor ou igual a n (teto).
double ceil(double n);
// Exemplos
ceil(1.3); // 2.0
ceil(1.5); // 2.0
ceil(1.7); // 2.0
ceil(2.0); // 2.0

// Produz o inteiro mais próximo de n (arredondamento).
double round(double n);
// Exemplos
round(1.3); // 1.0
round(1.5); // 2.0
round(1.7); // 2.0

// Calcula o seno de x (dado em radianos).
double sin(double x);
```

```
// Exemplos
sin(3.14); // 0.00159265

// Calcula a raiz quadrada de x.
double sqrt(double x);
// Exemplos
sqrt(2.0); // 1.41421

// Calcular x elevado a potência y.
double pow(double x, double y);
// Exemplos
pow(2.0, 4.0); // 16
```

3.2 Booleano

```
// Negação
// Produz true se x é false e produz false se x é true.
bool operator!(bool x);
// Exemplos
!(4 == 2 + 2); // false
!false; // true

// Conjunção (e)
// Produz true se a e b são true, false caso contrário.
bool operator&&(bool a, bool b);
// Exemplos
// par e maior que 12
int n = 14;
n % 2 == 0 && n > 12; // false

// Disjunção (ou)
// Produz true se a ou b é true, false caso contrário.
bool operator||(bool a, bool b);
// Exemplos
// menor de idade ou idoso
int idade = 72;
idade < 18 || idade >= 60; // true
```

3.3 String

Necessário incluir a biblioteca `string`.

```
// Concatenação
// Produz uma nova string juntado a string b no final da string a.
string operator+(string a, string b);
// Exemplos
string s = "Jorge";
"Ola " + s + "!"; // "Ola Jorge!"

// Substring
// Produz uma substring de this que começa na posição start e termina na
// posição min(len, start + len - 1).
// Requer que 0 <= start < this.length().
string substr(string this, int start, int len);
// Exemplos
string s = "jorge";
s.substr(1, 3); // "org"

// Número de bytes
```

```
// Produz o número de bytes de this.
int length(string this);
// Exemplos
string s = "jorge";
s.length(); // 5

// Repetição de caractere
// Produz uma string repetindo o caractere x n vezes.
int string(int n, char x);
// Exemplos
string s = string(3, 'a');
s.length(); // 3
s == "aaa"; // true
```

3.4 array

Necessário incluir a biblioteca `array`.

Operações básica

```
// Inicialização
array<int, 4> x = {6, 1, 8, 3}.

// Acesso ao elemento pelo índice, sem verificação
x[2]; // lê o valor da posição 2, que é 8.
x[3] = 4; // armazena o valor 4 na posição 3.
x[5]; // funcionamento indefinido, índice fora da faixa.

// Acesso ao elemento pelo índice, com verificação
x.at(0); // lê o valor da posição 0, que é 6.
x.at(1) = 7; // armazena o valor 7 na posição 1.
x.at(5); // erro em tempo de execução.
```

Exemplo

```
int main()
{
    array<int, 4> valores = {3, 1, 5, 2};
    int soma = 0;
    for (int v : valores) {
        soma = soma + v;
    }
    cout << soma << endl; // exibe 11
}
```

3.5 vector

Necessário incluir a biblioteca `vector`.

Operações básicas

```
// Todas as operações básica de array também funcionam com vector.

// Um vector pode ser inicializado com qualquer quantidade de elementos
vector<string> x = { "et", "telefone", "minha" };

// Quantidade de elementos
x.size(); // 3

// Adição de elemento no final do arranjo
```

```
x.push_back("casa"); // x = { "et", "telefone", "minha", "casa" }
x.size(); // 4

// Remoção do último elemento
x.pop_back(); // x = { "et", "telefone", "minha" }
```

Exemplo

```
int main()
{
    vector<int> valores = {3, 1, 5, 2};
    int soma = 0;
    for (int v : valores) {
        soma = soma + v;
    }
    cout << soma << endl; // exibe 11
}
```

3.6 set

Necessário incluir a biblioteca `set`.

Operações básicas

```
// Inicialização
set<int> s = {3, 1, 4};

// Inserção
s.insert(5);
s.insert(4);
s.insert(1); // s = {1, 5, 4, 3}
s.size() // 4

// Remoção
s.erase(10);
s.erase(1);
s.erase(3); // s = {4, 5}

// Verificação se um elemento está presente
s.count(4) // 1
s.count(3) // 0)
```

Exemplo

```
int main()
{
    set<int> valores = {1, 1, 5, 1};
    int soma = 0;
    for (int v : valores) {
        soma = soma + v;
    }
    cout << soma << endl; // exibe 6, só existe os elementos 1 e 5 no conjunto
}
```

4 Projeto de programas

Para projetar um programa seguimos os passos:

1. Análise

2. Definição dos tipos de dados
3. Especificação
4. Implementação
5. Verificação
6. Revisão

4.1 Análise

Determinar o problema que deve ser resolvido e as informações relevantes.

4.2 Definição dos tipos de dados

Identificar as informações e determinar como elas serão representadas.

4.3 Especificação

Escrever com mais precisão e com exemplos o que a função faz. Inclui

- Assinatura (nome da função, entradas e saída)
- Propósito
- Exemplos

4.4 Implementação

Escrever o corpo da função para que ela faça o que está na especificação.

Quando utilizar seleção?

Quando a forma de fazer a computação depende dos valores da entrada. Analisando os exemplos determinamos a forma da computação, se existir mais que uma forma, então precisamos usar seleção.

Quando utilize repetição?

Quando precisamos computar algo de forma incremental, repetindo um processo.

Em geral, se temos listas de valores na entrada, então precisaremos de repetição para processar as listas.

Se precisamos analisar todos os elementos de uma lista, um por vez, na ordem que eles aparecem, então usamos o "para cada". Se a forma de análise dos elementos é mais elaborada (precisamos do índice, precisamos analisar mais que um elemento por vez, etc), então utilizamos o "para".

Para escrever uma repetição com o "para cada", precisamos responder três perguntas:

1) Quais variáveis (valores) queremos calcular?
2) Como as variáveis são inicializadas?
3) Como as variáveis são atualizadas?

No caso do "para", também precisamos responder

1) Quais são as variáveis do laço e como elas são inicializadas?
2) Qual a condição do laço?
3) Como as variáveis do laço são atualizadas?

E o enquanto?

Em geral, utilizamos o "enquanto" para repetições que não envolvam lista de valores. Para projetar esse tipo de repetição, podemos começar com repetição física de código e generalizar para repetição lógica de código.

4.5 Verificação

Compilar e executar o programa para verificar se a implementação está de acordo com a especificação.

4.6 Revisão

Alterar a organização do programa para que fique mais fácil de ser lido, entendido e alterado.

4.7 Exemplos

Problema

O André viaja muito. Sempre antes de fazer uma viagem ele calcula o quanto ele irá gastar com combustível. Ele determina a distância que ele irá percorrer na viagem, o preço do litro do combustível e consulta as suas anotações para ver o consumo do carro, isto é, a quantidade de quilômetros que o carro anda com um litro de combustível e então faz o cálculo do custo. O André acha um pouco chato fazer os cálculos na mão, então ele pediu para você escrever um programa que faça os cálculos para ele.

Projeto

```c
#include "bscpp.hpp"

// Análise
// - Calcular o custo em reais para percorrer uma determinada distância levando
//   em consideração o desempenho   do carro e o preço do litro do combustível.
//
// Definição de tipos de dados
//
// - A distância em Km, rendimento em Km/l, preço em R$/l e o custo da viagem
//   em R$ serão representados por número positivos (double).

// Calcula o custo em reais para percorrer a distancia especificada
// considerando o rendimento do carro e o preco do litro do combustível.
double custo_viagem(double distancia, double rendimento, double preco) {
    return (distancia / rendimento) * preco;
}

examples {
    check_expect(custo_viagem(120.0, 10.0, 5.0), 60.0); // (120.0 / 10.0) * 5.0
    check_expect(custo_viagem(300.0, 15.0, 6.0), 120.0); // (300.0 / 15.0) * 6.0
}

int main() {
    run_tests();
}
```