C++ Para Programadores Java

Alisson Oliveira

Universidade Federal de Alagoas - UFAL
Campus Arapiraca

joe.alisson@gmail.com

22 de junho de 2013

Visão Geral

Hello World!!

C++

```
1  // Meu primeiro programa em c++
2  #include <iostream>
3  using namespace std;
4  int main() {
5     cout << "Hello World!";
6     return 0;
7  }
```

Java

```
1  package lp3;
2  // Meu primeiro programa em java
3  public class Main {
4      public static void main(String[] args) {
5          System.out.println("Hello World!");
6      }
7  }
```

Compilando

Para compilar basta executar esse comando no terminal.

Um arquivo

```
g++ source.cpp -o executavel
```

Vários arquivos

```
g++ source.cpp source1.cpp source2.cpp -o executavel
```

- g++ é o compilador
- source.cpp é o nome do arquivo
- executavel é o nome do aplicativo que o compilador criará

Declarando Váriaveis

Para criarmos programas úteis é necessário o uso de variáveis, para que seja possível guardar informações.

C++

```
1  int a(5);
2  float myNumber;
3  short x = 20;
4  unsigned short int y = 20;
5  signed long  z = -3000020;
```

Java

```
1  int a = 5;
2  float myNumber;
3  short x = 20;
```

Escopo de Variáveis

As variáveis devem ser declaradas antes de serem usadas. Podendo ser uma variável local ou global.

C++

```
1  #include <iostream>
2
3  using namespace std;
4
5  int x = 10; // variavel global
6
7  int main() {
8      // variaveis locais
9      int y = 20;
10     int z = x + y;
11     return 0;
12 }
```

Strings

C++ fornece suporte a string através da classe string.

C++

```cpp
#include <string>

int main() {
    string hi = "Hello World!";
    cout << hi;
}
```

Java

```java
public class Main {
    public static void main(String[] args) {
        String hi = "Hello World";
        System.out.println(hi);
    }
}
```

Literal

Constantes literais são usadas para expressar valores dentro do programa.

Tipo	Valor	Comentário
int	75	decimal
int	0113	octal
int	0x4b	hexadecimal
unsigned int	75u	
long	75l	
unsigned long	75ul	
double	3.14159	
float	3.1334f	
float	6.02e23	$6.02x10^{23}$
char	'a'	
string	"Hello"	
bool	true	

Constantes Definidas e Declaradas

São variáveis que não podem ter seus valores alterados.

C++

```
1  #define PI 3.14159 // constante definida
2  const int raio = 2; //constante declarada
3  int main() {
4      const string msg = "a area do circulo:";
5      cout << msg << endl << PI*raio*raio;
6      return 0;
7  }
```

Java

```
1  public static void main(String[] args) {
2      final double PI = 3.14159;
3      final int RAIO = 2;
4      final String msg = "o raio do circulo:";
5      System.out.println(msg + "\n" + (PI*RAIO*RAIO));
6  }
```

Atribuição (=)

Atribui um valor à variáveis.

C++

```cpp
int main() {
    int a = 10, b = 4, c, d, e;
    d = e = 2;
    d += a + (c = b * e);
    cout << "a: " << a << endl << "b: " << b << endl;
    cout << "c: " << c << endl << "d: " << d << endl;
    cout << "e: " << e;
}
```

Java

```java
public static void main(String[] args) {
    int a = 10, b = 4, c, d, e;
    d = e = 2;
    d += a + (c = b * e);
    System.out.println("a: "+ a +"\nb: "+ b + "\nc: " +
    c + "\nd: " + d + "\ne: " + e);
}
```

Aritméticos (+, -, *, /, %)

Usados para operações aritméticas.

C++

```cpp
int main() {
    int a = 10, b;
    b = a * 2  % 3;
    a = a * 5 / b % 4;
    cout << "a: " << a << endl << "b: " << b << endl;
}
```

Java

```java
public static void main(String[] args) {
    int a = 10, b;
    b = a * 2 % 3;
    a = a * 5 / b % 4;
    System.out.println("a: " + a + "\nb: " + b);
}
```

Relacionais e Lógicos

Usados para fazer comparações e criar sentenças lógicas. O retorno desses operadores é um valor booleano.

Operador	Descrição
==	Igual a
! =	Diferente
>	Maior que
<	Menor que
>=	Maior ou igual a
<=	Menor ou igual a
!	Negação (NOT)
||	OU (OR)
&&	E (AND)

Tabela: Operadores Relacionais e Lógicos

Condicional (? :) e Virgula (,)

- (? :): Retorna um valor de acordo com o valor da expressão avaliada.
- (,): Usado para separar duas ou mais expressões onde apenas uma é esperada.

C++

```
1  int main() {
2      int a=2, b=7, c;
3      c = (a > b) ? a : b; // operador ? :
4      c = (b = c, b*a); // operador ,
5      cout << c;
6  }
```

Java

```
1  public static void main(String[] args) {
2      int a=2, b=7, c;
3      c = (a > b) ? a : b; // operador ? :
4      c*=a;
5      System.out.println(c);
6  }
```

sizeof

Retorna o tamanho em bytes de um tipo ou de uma variável.

C++

```
1  int main() {
2      int a, b;
3      a = sizeof (char);
4      b = sizeof (b);
5      cout << a << endl;
6      cout << b << endl;
7      return 0;
8  }
```

Saída Padrão (cout)

A saída padrão de um programa é a tela, cout é o objeto que tem acesso a ela.

C++

```
1  int main() {
2       string nome = "juca";
3       int idade = 20;
4       cout << "Oi, eu sou " << nome << endl;
5       cout << "Tenho " << idade << " anos";
6  }
```

Java

```
1  public static void main(String[] args) {
2      String nome = "juca";
3      int idade = 20;
4      System.out.println("Oi, eu sou " + nome);
5      System.out.println("Tenho " + idade + " anos");
6  }
```

Entrada Padrão (cin)

C++

```cpp
int main() {
    int x;
    float y;
    cout << "Entre com dois numeros: " << endl;
    cin >> x >> y;
    cout << "a soma dos numeros: " << x+y << endl;
    return 0;
}
```

Java

```java
public static void main(String[] args) {
    int x;
    float y;
    Scanner sc = new Scanner(System.in);
    System.out.println("Entre com dois numeros:");
    x = sc.nextInt();
    y = sc.nextFloat();
    System.out.println("a soma dos numeros: " + (x+y));
}
```

cin e strings

É recomendado o uso da função getline quando a entrada for uma string.

C++

```cpp
int main() {
    string nome;
    cout << "Qual o seu nome? ";
    getline (cin, nome);
    cout << "Oi " << nome << endl;
    return 0;
}
```

Java

```java
public static void main(String[] args) {
    String nome;
    Scanner sc = new Scanner(System.in);
    System.out.println("Qual o seu nome? ");
    nome = sc.nextLine();
    System.out.println("Oi " + nome);
}
```

stringstream

A classe stringstream, definida no arquivo sstream é especialmente útil na conversão de string para valores númericos e vice versa.

C++

```cpp
#include <sstream>
int main() {
    string str = "3.00", q = "2";
    float preco; int qnt;
    stringstream(str) >> preco;
    stringstream(q) >> qnt;
    cout << preco * qnt;
}
```

Java

```java
public static void main(String[] args) {
    String str = "3.00", q = "2";
    float preco = Float.parseFloat(str);
    int qnt = Integer.parseInt(q);
    System.out.println(preco * qnt);
}
```

Estruturas de Controle

C++ fornece estruturas para determinar o comportamento do programa de acordo com alguma circustância.

- (switch, if e else) Permite fazer uma seleção dos comandos a serem executados
- (while, do while e for) Repete um trecho de código até uma condição ser satisfeita.
- (goto) Salta para um determinado ponto no programa.
- (funções) Permite agrupar blocos de comandos que possuem comportamento parecido.

if e else

C++

```
1  int main() {
2      if (x > 0)
3          cout << "positivo";
4      else if (x < 0)
5          cout << "negativo";
6      else
7          cout << "zero";
8      return 0;
9  }
```

Java

```
1  public static void main(String[] args) {
2      if (x > 0)
3          System.out.println("positivo");
4      else if (x < 0)
5          System.out.println("negativo");
6      else
7          System.out.println("zero");
8  }
```

while

Permite a repetição até que a condição dada seja avaliada como falsa.

C++

```
1  int main() {
2      int n = 5;
3      while (n>0) {
4          cout << n-- << ", ";
5      }
6      cout << "Fire!";
7  }
```

Java

```
1  public static void main(String[] args) {
2      int n = 5;
3      while (n>0) {
4          System.out.print(n-- + ", ");
5      }
6      System.out.println("Fire!");
7  }
```

do while

C++

```
1  int n;
2  do {
3      cout << "digite um numero (0 para terminar): ";
4      cin >> n;
5      cout << "numero dado: " << n << endl;
6  } while (n != 0);
```

Java

```
1  int n;
2  do {
3      System.out.println("digite um numero (0 para terminar): ");
4      n = sc.nextInt();
5      System.out.println("numero dado: " + n);
6  } while (n != 0);
```

for

C++

```
1  int main() {
2      for (int n=10; n>0; n--) {
3          cout << n << ", ";
4      }
5      cout << "FIRE!\n";
6      return 0;
7  }
```

Java

```
1  public static void main(String[] args) {
2      for (int n=10; n>0; n--) {
3          System.out.print(n + ", ");
4      }
5      System.out.println("FIRE!");
6  }
```

break

Para a repetição.

C++

```cpp
for (int n = 10; n > 0; n--) {
    cout << n << ", ";
    if (n == 3) {
        cout << "contagem abortada!";
        break;
    }
}
```

Java

```java
for (int n = 10; n > 0; n--) {
    System.out.print(n + ", ");
    if (n == 3) {
        System.out.println("contagem abortada!");
        break;
    }
}
```

continue

Para a iteração sendo executada e pula para próxima iteração.

C++

```
1  for (int n = 10; n > 0; n--) {
2      if (n == 5)
3          continue;
4      cout << n << ", ";
5  }
6  cout << "FIRE!\n";
```

Java

```
1  for (int n = 10; n > 0; n--) {
2      if (n == 5)
3          continue;
4      System.out.print(n + ", ");
5  }
6  System.out.println("FIRE!");
```

goto

Faz um salto para um label.

C++

```
1  for(int i=0; i < 10; i++) {
2      for(int j=0; j < 10; j++) {
3          if(i==5 && j==6)  goto out;
4          cout << i << j << endl;
5      }
6  }
7  out:
```

Java

```
1  out:
2      for (int i = 0; i < 10; i++) {
3          for (int j = 0; j < 10; j++) {
4              if (i == 5 && j == 6)  break out;
5              System.out.println(i + "" + j);
6          }
7      }
```

switch

C++

```
1  int x = 4;
2  switch (x) {
3      case 1: cout << "1"; break;
4      case 2:
5      case 3: cout << "2 ou 3"; break;
6      default: cout << "nem 1, 2 ou 3";
7  }
```

Java

```
1  int x = 4;
2  switch (x) {
3      case 1: System.out.println("1"); break;
4      case 2:
5      case 3: System.out.println("2 ou 3"); break;
6      default: System.out.println("nem 1, 2 ou 3");
7  }
```

Declarando Funções

Função é um conjunto de comandos que são executados quando ela é chamada em algum ponto do programa.

C++

```
1  int add(int x, int y) {
2      return x + y;
3  }
4
5  int main() {
6      cout << add(3, 4);
7  }
```

Java

```
1  static int add(int x, int y) {
2      return x + y;
3  }
4  public static void main(String[] args) {
5      System.out.println(add(3,4));
6  }
```

Parâmetro Com Valor Padrão

Podemos especificar um valor padrão para os últimos parâmetros de uma função.

C++

```
1  int add(int x, int y=1) {
2      return x + y;
3  }
4
5  int main() {
6      cout << add(3) << endl;
7      cout << add(3, 8);
8  }
```

Sobrecarga de Funções

Funções podem ter o mesmo nome, desde que tenham parâmetros diferentes.

C++

```
1  int op(int x, int y) {  return x + y; }
2  float op(float x, float y) { return x * y; }
3
4  int main() {
5      cout << op(3, 4) << endl;
6      cout << op(4.0f, 5.0f);
7  }
```

Java

```
1  static int op(int x, int y) { return x + y; }
2  static float op(float x, float y) { return x * y; }
3
4  public static void main(String[] args) {
5      System.out.println(op(3,4));
6      System.out.println(op(4.0f,5.0f));
7  }
```

Funções inline

Esse tipo de função indica ao compilador que uma substituição em linha é preferível que o o mecanismo padrão.

C++

```
1  inline int add(int x, int y) {
2      return x + y;
3  }
4
5  int main() {
6      cout << add(3, 4) << endl;
7  }
```

Protótipo de Função

Permite o uso da função antes de sua declaração.

C++

```
1
2  int add(int, int);
3  float mult(float x, float y);
4
5  int main() {
6      cout << add(3, 4) << endl;
7      cout << mult(3, 4);
8  }
9
10 float mult(float x, float y) {
11     return x * y;
12 }
13
14 int add(int x, int y) {
15     return x + y;
16 }
```

Declarando Array

Array é um conjunto de elementos do mesmo tipo que podem ser referenciados individualmente.

C++

```cpp
using namespace std;

int main() {
    int array[] = {32, 223, 12, 3 , 2};
    char chars[5];
    return 0;
}
```

Java

```java
public static void main(String[] args) {
    int array[] = {32, 223, 12, 3 , 2};
    char chars[] =  new char[5];
    System.out.println(array + " " + chars);
}
```

Acessando Elementos do Array

C++

```cpp
int main() {
    int ints [] = { 16, 2, 77, 40, 12071 };
    int n, result = 0;
    for (n = 0; n < 5; n++) {
        result += ints[n];
    }
    cout << result;
    return 0;
}
```

Java

```java
public static void main(String[] args) {
    int ints [] = { 16, 2, 77, 40, 12071 };
    int n, result = 0;
    for (n = 0; n < 5; n++) {
        result += ints[n];
    }
    System.out.println(result);
}
```

Array Multidimensional

Array Multidimensional é um array de arrays.

C++

```cpp
int main() {
    int matriz[10][10];
    for (int i = 0; i < 10; i++) {
        for(int j = 0; j < 10; j++) {
            matriz[i][j] = i*j;
        }
    }
}
```

Java

```java
public static void main(String[] args) {
    int matriz[][] = new int[10][10];
    for (int i = 0; i < 10; i++) {
        for(int j = 0; j < 10; j++) {
            matriz[i][j] = i*j;
        }
    }
```

Array Como Parâmetro

A passagem do array como parâmetro é feita por referência.

C++

```
1  void printarray(int arg[], int length) {
2      for (int n = 0; n < length; n++) cout << arg[n] << " ";
3  }
4  int main() {
5      int array[3] = {2,4,5};
6      printarray(array, 3);
7  }
```

Java

```
1  static void printarray(int arg[], int length) {
2      for (int n = 0; n < length; n++)
3          System.out.print(arg[n] + " ");
4  }
5  public static void main(String[] args) {
6      int array[] = new int[]{2,4,5};
7      printarray(array, 3);
8  }
```

Declarando variáveis do tipo ponteiro

Ponteiro é um tipo de dado que guarda o endereço da memória (referência) de uma variável ou objeto.

C++

```
1  int main() {
2      int * numero;
3      char * caractere;
4      float * real
5      return 0;
6  }
```

Operador de Referência (&)

O operador de referência retorna o endereço de memória da variável, o qual é lido como "endereço de".

C++

```
1  int main() {
2      int valor;
3      int * ponteiro;
4      ponteiro = &valor;
5      cout << ponteiro;
6      return 0;
7  }
```

Operador de Desreferencia (*)

Usando ponteiro podemos acessar diretamente o valor guardado em uma variável.

C++

```
1  int main() {
2      int valor1, valor2;
3      int * ponteiro;
4      ponteiro = &valor1;
5      *ponteiro = 10;
6      ponteiro = &valor2;
7      *ponteiro = 20;
8      cout << "valor1: " << valor1 << endl;
9      cout << "valor2: " << valor2 << endl;
10     return 0;
11 }
```

Aritmética de Ponteiros e Arrays

O conceito de array é próximo para o de ponteiro. O identificador de um array é, de fato, o endereço para o primeiro elemento.

C++

```
1  int main () {
2      int numeros[5];
3      int * p;
4      p = numeros;
5      *p = 10;
6      p++;
7      *p = 20;
8      p = &numeros[2];
9      *p = 30;
10     p = numeros + 3;
11     *p = 40;
12     p = numeros;
13     *(p+4) = 50;
14     for (int n=0; n<5; n++)
15         cout << numeros[n] << ", ";
16     return 0;
17 }
```

Ponteiros Para Ponteiros

Um ponteiro pode apontar pra outro ponteiro, o segundo podendo apontar pra um valor ou até mesmo pra outro ponteiro. Para cada nível de referência adicionamos um asterisco (*) na declaração do ponteiro.

C++

```
1  int main () {
2        char a;
3        char * b;
4        char ** c;
5        a = 'z';
6        b = &a;
7        c = &b;
8        cout << "a: " << a << endl;
9        cout << "*b: " << *b << endl;
10       cout << "**c: " << **c << endl;
11 }
```

Ponteiros void

É um tipo especial de ponteiro. void representa a falta de tipo.

C++

```
1  void add(void* data, int psize) {
2      if (psize == sizeof(char)) {
3          char* pchar;
4          pchar = (char*) data;
5          ++(*pchar);
6      } else if (psize == sizeof(int)) {
7          int* pint;
8          pint = (int*) data;
9          ++(*pint);
10     }
11 }
12
13 int main() {
14     char a = 'x';
15     int b = 1602;
16     add(&a, sizeof(a));
17     add(&b, sizeof(b));
18     cout << a << ", " << b << endl;
19 }
```

Ponteiros para funções

C++

```
1  int add(int a, int b) {
2      return (a + b);
3  }
4  int sub(int a, int b) {
5      return (a - b);
6  }
7  int op(int x, int y, int (*func)(int, int)) {
8      int g;
9      g = (*func)(x, y);
10     return (g);
11 }
12
13 int main() {
14     int m, n;
15     int (*minus)(int, int) = sub;
16     m = op(7, 5, add);
17     n = op(20, m, minus);
18     cout << n;
19     return 0;
20 }
```

Alocação Dinâmica

Até agora, todos os exemplos dados, tem a memória disponível para alocar as variáveis declaradas. Tendo o tamanho determinado no source code, antes da execução do programa.
Mas a maioria dos programas precisam alocar memória durante o tempo de execução. Para isso usamos alocação dinâmica de memória.

Operadores new e new[]

Usados para solicitar memória em tempo de execução.

C++

```
1  int main() {
2      int * array = new int[5];
3      int * x = new int;
4      cout << *x << endl << array[*x];
5      return 0;
6  }
```

Java

```
1  public static void main(String[] args) {
2      int[] array = new int[5];
3      int x = new Integer(0); // int x = 0;
4      System.out.println(x + "\n" + array[x]);
5  }
```

Operadores delete e delete[]

Libera a memória que foi dinamicamente alocada.

C++

```
1  int main() {
2      int i, n;
3      int * p;
4      cout << "Quantos numeros vai digitar ?";
5      cin >> i;
6      p = new (nothrow) int[i];
7      if (p == 0)
8          cout << "Error: memoria nao alocada";
9      else {
10         for (n = 0; n < i; n++) {
11             cout << "digite o numero: ";
12             cin >> p[n];
13         }
14         cout << "numeros digitados: ";
15         for (n = 0; n < i; n++)
16             cout << p[n] << ", ";
17         delete[] p;
18     }
19 }
```

Estruturas de Dados

É um conjunto de dados agrupados em um identificador.
Esses elementos podem ter tipos e tamanhos diferentes, são chamados de membros.

struct

C++

```
1  struct filme {
2      string titulo;
3      int ano;
4  } fil;
5
6  void printfilme(filme fil) {
7      cout << fil.titulo;
8      cout << " (" << fil.ano << ")\n";
9  }
10
11 int main() {
12     string mystr;
13     cout << "Digite o titulo: ";
14     getline(cin, fil.titulo);
15     cout << "Digite o ano: ";
16     getline(cin, mystr);
17     stringstream(mystr) >> fil.ano;
18     cout << "meu filme favorito:\n";
19     printfilme(fil);
20     return 0;
21 }
```

Ponteiros Para Estruturas

C++

```
1  struct filme {
2      string titulo;
3      int ano;
4  };
5
6  int main() {
7      string mystr;
8      filme fil;
9      filme * pfilme;
10     pfilme = &fil;
11     cout << "Digite o titulo: ";
12     getline(cin, pfilme->titulo);
13     cout << "Digite o ano: ";
14     getline(cin, mystr);
15     (stringstream) mystr >> pfilme->ano;
16     cout << "\nVoce Digitou:\n";
17     cout << pfilme->titulo;
18     cout << " (" << pfilme->ano << ")\n";
19     return 0;
20 }
```

Tipo de Dado definido (typedef)

Permite a definição de novos tipos baseado em tipos já existentes.

C++

```
1  typedef char C;
2  typedef unsigned int WORD;
3  typedef char * pChar;
4  typedef char field [50];
5
6  int main() {
7      C caractere = 'a';
8      WORD num = caractere;
9      pChar d = &caractere;
10     field f;
11     f[0] = *d;
12     cout << f[0] << " : " << num;
13 }
```

Union

Permite acessar diferentes tipos de dados em um mesmo endereço de memória.

C++

```
1  union ascii {
2      char c;
3      int i;
4  } var;
5
6  int main() {
7      var.c = 'a';
8      ascii var2;
9      var2.i = var.i + 11;
10     cout << var.c <<  " : "<<  var.i << endl;
11     cout << var2.c <<  " : "<<  var2.i << endl;
12 }
```

Enumeration

Cria um novo tipo de dado que pode conter algo diferente do tipos de dados básicos.

C++

```
1
2  enum colors {black, blue, green, cyan, red, purple,
3      yellow, white};
4
5  int main() {
6      colors cor;
7      cor = blue;
8      if (cor == green)
9          cor = red;
10     return 0;
11 }
```

Classes

Classe é um conceito extendido de estruturas de dados. Podemos declarar funções e dados (atributos).

C++

```
1  class Retangulo {
2      int x, y;
3  public:
4      void set_valores(int, int);
5      int area() { return (x * y); }
6  };
7  void Retangulo::set_valores(int a, int b) {
8      x = a;
9      y = b;
10 }
11 int main() {
12     Retangulo rect, rectb;
13     rect.set_valores(3, 4);
14     rectb.set_valores(5, 6);
15     cout << "rect area: " << rect.area() << endl;
16     cout << "rectb area: " << rectb.area() << endl;
17 }
```

Classes

Java

```
1  public class Retangulo {
2      private int x;
3      private int y;
4      public void setValores(int a, int b) {
5          x = a;
6          y = b;
7      }
8      public int area() {
9          return x * y;
10     }
11
12     public static void main(String[] args) {
13         Retangulo rect = new Retangulo();
14         Retangulo rectb = new Retangulo();
15         rect.setValores(3, 4);
16         rectb.setValores(5, 6);
17         System.out.println("rect area: " + rect.area());
18         System.out.println("rectb area: " + rectb.area());
19     }
20 }
```

Construtores

O Construtor é chamado automaticamente na criação de um objeto.

C++

```
1  class Retangulo {
2      int largura, altura;
3  public:
4      Retangulo(int, int);
5      int area() {
6          return (largura * altura);
7      }
8  };
9  Retangulo::Retangulo(int a, int b) {
10     largura = a;
11     altura = b;
12 }
13 int main() {
14     Retangulo rect(3, 4);
15     Retangulo rectb(5, 6);
16     cout << "rect area: " << rect.area() << endl;
17     cout << "rectb area: " << rectb.area() << endl;
18     return 0;
19 }
```

Construtores

Java

```
1  public class Retangulo {
2      private int x;
3      private int y;
4
5      public Retangulo(int a, int b) {
6          x = a;
7          y = b;
8      }
9
10     public int area() {
11         return x * y;
12     }
13
14     public static void main(String[] args) {
15         Retangulo rect = new Retangulo(3, 4);
16         Retangulo rectb = new Retangulo(5, 6);
17         System.out.println("rect area: " + rect.area());
18         System.out.println("rectb area: " + rectb.area());
19     }
20 }
```

Destrutores

São chamados automaticamente quando o objeto é destruido

C++

```
1  class CRectangle {
2      int *width, *height;
3  public:
4      CRectangle(int, int);
5      ~CRectangle();
6      int area() {
7          return (*width * *height);
8      }
9  };
10 CRectangle::CRectangle(int a, int b) {
11     width = new int;
12     height = new int;
13     *width = a;
14     *height = b;
15 }
16 CRectangle::~CRectangle() {
17     delete width;
18     delete height;
19 }
```

Ponteiros Para Classes

C++

```
1  class Retangulo {
2      int largura, altura;
3  public:
4      void set_values(int, int);
5      int area(void) {  return (largura * altura); }
6  };
7  void Retangulo::set_values (int a, int b) {
8      largura = a;
9      altura = b;
10 }
11 int main() {
12     Retangulo a, *b;
13     b = new Retangulo;
14     a.set_values(1, 2);
15     b->set_values(3, 4);
16     cout << "a area: " << a.area() << endl;
17     cout << "*b area: " << b->area() << endl;
18     delete b;
19     return 0;
20 }
```

Sobrecarga de Operadores

C++ Permite o uso de operadores padrões para executar operações com classes. Para isso é necessário definirmos metodos que sobreescrevam o comportamento padrão do operador.

Operadores Sobrecarregaveis

```
+   -   *   /   %   +=   -=   *=   /=   %=   ==   !=   <   >   <=   >=
<<=   >>=   ~   &=   ^=   |=   =   ++   --   !   &&   ||   &   |   ^
<<   >>   []   ()   ,   ->*   ->   new   new[]   delete   delete[]
```

Sobrecarga de Operadores

C++

```
1  class Vetor {
2  public:
3      int x, y;
4      Vetor() {
5          x = 1, y = 1;
6      };
7      Vetor operator +(Vetor);
8  };
9  Vetor Vetor::operator+(Vetor param) {
10     Vetor temp;
11     temp.x = x + param.x;
12     temp.y = y + param.y;
13     return (temp);
14 }
15
16 int main() {
17     Vetor a, b, c;
18     c = a + b;
19     cout << "(" << c.x << "," << c.y << ")";
20     return 0;
21 }
```

this

Representa um ponteiro para o objeto que está executando o método.

C++

```
1  class Dummy {
2  public:
3      bool igual(Dummy& param);
4  };
5
6  bool Dummy::igual(Dummy& param) {
7      return &param == this;
8  }
9
10 int main() {
11     Dummy a;
12     Dummy* b = &a;
13     if (b->igual(a))
14         cout << " &a = b";
15     return 0;
16 }
```

this

Java

```
1  public class Dummy {
2
3      public boolean igual(Dummy param) {
4          return this.equals(param);
5      }
6
7      public static void main(String[] args) {
8          Dummy a = new Dummy();
9          Dummy b = a;
10         if (b.igual(a))
11             System.out.println("a = b");
12     }
13 }
```

Membros static

São membros da Classe. Eles têm o mesmo valor para todos os objetos da classe.

C++

```
1  class Dummy {
2  public:
3      static int n;
4      Dummy() { n++; };
5      ~Dummy() { n--; };
6      static void destroy(Dummy& c) { delete &c; }
7  };
8  int Dummy::n = 0;
9  int main() {
10     Dummy a;
11     Dummy b;
12     Dummy * c = new Dummy;
13     cout << a.n << endl;
14     Dummy::destroy(*c);
15     cout << Dummy::n << endl;
16     return 0;
17 }
```

Membros static

Java

```
1  public class Dummy {
2      public static int n = 0;
3
4      public Dummy() {
5          n++;
6      }
7
8      static void destroy(Dummy c) {
9          n--;
10     }
11
12     public static void main(String[] args) {
13         Dummy a = new Dummy();
14         Dummy b = new Dummy();
15         Dummy c = new Dummy();
16         System.out.println(a.n);
17         Dummy.destroy(c);
18         System.out.println(Dummy.n);
19     }
20 }
```

Funções friend

São funções especiais que têm acesso aos membros privados de uma classe.

C++

```
1  class Retangulo {
2      int largura, altura;
3  public:
4      Retangulo() {};
5      Retangulo(int x, int y) { largura = x, altura = y; }
6      int area () { return (largura * altura); }
7      friend Retangulo duplicar(Retangulo);
8  };
9  Retangulo duplicar(Retangulo rect) {
10     Retangulo rectres;
11     rectres.largura = rect.largura * 2;
12     rectres.altura = rect.altura * 2;
13     return (rectres);
14 }
15 int main() {
16     Retangulo rect(2,3), rectb;
17     rectb = duplicar(rect);
18     cout << rectb.area();
19 }
```

Classes friend

C++

```
1  class Quadrado;
2
3  class Retangulo {
4      int largura, altura;
5  public:
6      int area() { return (largura * altura); }
7      void convert(Quadrado a);
8  };
9
10 class Quadrado {
11 private:
12     int lado;
13 public:
14     void setLado(int a) { lado = a; }
15     friend class Retangulo;
16 };
17
18 void Retangulo::convert(Quadrado a) {
19     largura = a.lado;
20     altura = a.lado;
21 }
```

Herança

Permite a criação de classes derivadas de outras classes.

C++

```
1  class Poligono {
2  protected:
3      int largura, altura;
4  public:
5      void set_values(int a, int b) { largura = a; altura = b; }
6  };
7
8  class Retangulo: public Poligono {
9  public:
10     int area() { return (largura * altura); }
11 };
12
13 int main() {
14     Retangulo rect;
15     rect.set_values(4, 5);
16     cout << rect.area() << endl;
17     return 0;
18 }
```

Herança

Java

```
1  public class Poligono {
2      protected int largura, altura;
3
4      public void setValues(int a, int b) {
5          largura = a;
6          altura = b;
7      }
8
9      public static void main(String[] args) {
10         Retangulo rect = new Retangulo();
11         rect.setValues(4, 5);
12         System.out.println(rect.area());
13     }
14 }
15
16 class Retangulo extends Poligono {
17     public int area() {
18         return largura * altura;
19     }
20 }
```

O que não é Herdado

C++

```
1  class Mae {
2  public:
3      Mae() { cout << "Mae: sem parametros\n"; }
4      Mae(int a) { cout << "Mae: com parametro\n"; }
5  };
6
7  class Filha: public Mae {
8  public:
9      Filha(int a) { cout << "Filha: com parametro\n\n"; }
10 };
11
12 class Filho: public Mae {
13 public:
14     Filho(int a) : Mae(a) { cout << "Filho: com parametro\n\n"; }
15 };
16
17 int main() {
18     Filha cynthia(0);
19     Filho daniel(0);
20     return 0;
21 }
```

Herança Múltipla

Em c++ é possível uma classe ter mais de uma superclasse.

C++

```
1  class CPolygon {
2  protected:
3      int width, height;
4  public:
5      void set_values (int a, int b) { width=a; height=b;}
6  };
7  class COutput {
8  public:
9      void output(int i);
10 };
11 void COutput::output(int i) {
12     cout << i << endl;
13 }
14 class CRectangle: public CPolygon, public COutput {
15 public:
16     int area() {  return (width * height); }
17 };
```

Ponteiros para Classe Base

Ponteiro para uma classe derivada é compatível com a classe base.

C++

```
1  class Poligono {
2  protected:
3      int largura, altura;
4  public:
5      void set_values(int a, int b) {
6          largura = a;  altura = b;
7      }
8  };
9  class Retangulo: public Poligono {
10 public:
11     int area() { return (largura * altura); }
12 };
13 int main() {
14     Retangulo rect;
15     Poligono * poli = &rect;
16     poli->set_values(4, 5);
17     cout << rect.area() << endl;
18     return 0;
19 }
```

Membros virtual

Um membro de uma classe que pode ser redefinido em uma classe derivada é um membro virtual.

C++

```
1  class Polygon {
2  protected:
3      int largura, altura;
4  public:
5      void set_values(int a, int b) {
6          largura = a;
7          altura = b;
8      }
9      virtual int area() {
10         return (0);
11     }
12 };
13 class Retangulo: public Polygon {
14 public:
15     int area() {
16         return (largura * altura);
17     }
18 };
```

Membros virtual

C++

```
1  int main() {
2      Retangulo rect;
3      Polygon poly;
4      Polygon * ppoly1 = &rect;
5      Polygon * ppoly3 = &poly;
6      ppoly1->set_values(4, 5);
7      ppoly3->set_values(4, 5);
8      cout << ppoly1->area() << endl;
9      cout << ppoly3->area() << endl;
10     return 0;
11 }
```

Classes Abstratas

Em uma classe abstrata deve haver pelo menos uma função sem implementação. Isso é feito igualando uma função virtual a 0. Essa função é chamada de função virtual pura.

C++

```
1  class Poligono {
2  protected:
3      int largura, altura;
4  public:
5      virtual ~Poligono(){};
6      void set_values(int a, int b) {
7          largura = a;  altura = b;
8      }
9      virtual int area(void) =0;
10     void printarea(void) { cout << this->area() << endl; }
11 };
12
13 class Retangulo: public Poligono {
14 public:
15     int area(void) { return (largura * altura); }
16 };
```

Classes Abstratas

C++

```
1  int main() {
2      Poligono * ppoly1 = new Retangulo;
3      ppoly1->set_values(4, 5);
4      ppoly1->printarea();
5      delete ppoly1;
6      return 0;
7  }
```

Funções Template

Permite a criação de funções que podem ser adaptadas para mais de um tipo.

C++

```
1  template <class T> T Maior(T a, T b) {
2      T result;
3      result = (a > b) ? a : b;
4      return result;
5  }
6
7  int main() {
8      int i = 5, j = 6, k;
9      long l = 10, m = 5, n;
10     k = Maior<int>(i, j);
11     n = Maior(l, m);
12     cout << k << endl;
13     cout << n << endl;
14     return 0;
15 }
```

Classes Template

Possibilita a definição de classes genéricas.

C++

```
1  template<class T> class Par {
2      T a, b;
3  public:
4      Par(T primeiro, T segundo) {
5          a = primeiro;  b = segundo;
6      }
7      T maior();
8  };
9  template<class T> T Par<T>::maior() {
10     T retval;
11     retval = a > b ? a : b;
12     return retval;
13 }
14 int main() {
15     Par<int> object(100, 75);
16     Par<long> object2(100000, 75532);
17     cout << object.maior() << endl;
18     cout << object2.maior();
19 }
```

Especialização de Template

C++

```
1  template<class T> class Container {
2    T elemento;
3  public:
4    Container(T arg) { elemento = arg; }
5    T aumentar() { return ++elemento; }
6  };
7  template<> class Container<char> {
8    char elemento;
9  public:
10   Container(char arg) { elemento = arg; }
11   char aumentar() {
12     if (elemento >= 'a' && elemento <= 'z')elemento += 'A'-'a';
13     return elemento;
14   }
15 };
16 int main() {
17   Container<int> myint(10);
18   Container<char> mychar('k');
19   cout << myint.aumentar() << endl;
20   cout << mychar.aumentar() << endl;
21 }
```

Namespace

Permite agrupar entidades em um escopo. Desse modo podemos dividir um escopo em "sub-escopos".

C++

```
1  namespace first {
2      int var = 5;
3  }
4  namespace second {
5      double var = 3.1416;
6  }
7  int main() {
8      cout << first::var << endl;
9      cout << second::var << endl;
10     return 0;
11 }
```

using

Especifica qual namespace será usado em um determinado escopo.

C++

```
1  namespace first {
2      int x = 5;
3      int y = 10;
4  }
5  namespace second {
6      double x = 3.1416;
7      double y = 2.7183;
8  }
9  int main() {
10     using first::x;
11     using second::y;
12     cout << x << endl;
13     cout << y << endl;
14     cout << first::y << endl;
15      cout << second::x << endl;
16      return 0;
17 }
```

Exceções

Permite reagir a circunstância inesperadas como erros de runtime.

C++

```
1  int main() {
2      int x = 7;
3      string str = "string exception";
4      try {
5          switch(x) {
6          case 0:
7              throw str;
8          case 1:
9              throw 20;
10         default:
11             throw 3.2;
12         }
13     } catch (int e) {
14         cout << "Exception " << e << endl;
15     } catch (string& e) {
16         cout << "Exception " << e << endl;
17     } catch (...) {
18         cout << "An exception occurred. ";
19     }
```

Criando Exceções

C++

```
1  #include <exception>
2
3  using namespace std;
4  class myexception: public exception {
5      virtual const char* what() const throw() {
6          return "My exception happened";
7      }
8  }myex;
9
10 int main() {
11     try {
12         throw myex;
13     } catch (exception& e) {
14         cout << e.what() << endl;
15     }
16     return 0;
17 }
```

Conversão implícita

Conversão implícita não requer um operador. Acontece quando um valor é copiado para um tipo compatível.

C++

```
1  class A {};
2  class B {
3  public:
4      B(A a) { }
5      void print(int x) {
6          cout << x;
7      };
8  };
9
10 int main() {
11     short x = 200;
12     int y = x;
13     A a;
14     B b = a;
15     b.print(y);
16 }
```

Conversão explícita

Em alguns casos é necessário fazer uma conversão explícita.

C++

```
1  class A {};
2  class B {
3  public:
4      B(A a) { }
5      void print(int x) {
6          cout << x << endl;
7      };
8  };
9
10 int main() {
11     short x = 200;
12     int y = (int) x;
13     long z = long(y);
14     A a;
15     B b = (B)a;
16     b.print(y);
17     b.print(z);
18 }
```

dynamic_cast

Usado para conversão entre ponteiros e referências para objetos.

C++

```
1  class CBase {
2      virtual void dummy() { }
3  };
4  class CDerived: public CBase {
5      int a;
6  };
7  int main() {
8      try {
9          CBase * pba = new CDerived;
10         CBase * pbb = new CBase;
11         CDerived * pd;
12         pd = dynamic_cast<CDerived*>(pba);
13         if (pd == 0)
14               cout << "Null pointer on first type-cast" << endl;
15         pd = dynamic_cast<CDerived*>(pbb);
16         if (pd == 0)
17              cout << "Null pointer on second type-cast" << endl;
18     } catch (exception& e) {
19         cout << "Exception: " << e.what();
```

static_cast

Usado para conversão entre classes relacionadas.

C++

```
1  class CBase {
2  public:
3      int x;
4  };
5  class CDerived: public CBase {
6  public:
7      int y;
8  };
9
10 int main() {
11     CBase * a = new CBase;
12     a->x = 20;
13     CDerived * b = static_cast<CDerived*>(a);
14     b->y = 10;
15     cout << b->y << " " << b->x;
16     return 0;
17 }
```

const_cast

Usado para conversão de contantes.

C++

```
1  void print(char * str) {
2      cout << str << endl;
3  }
4
5  void print2(const char * str) {
6      cout << str << endl;
7  }
8
9  int main() {
10     const char * c = "constant";
11     char *d = "no-constant";
12     print(const_cast<char *>(c));
13     print2(const_cast<const char *>(d));
14     return 0;
15 }
```

typeid

Permite checar o tipo de um objeto.

C++

```
1  #include <typeinfo>
2
3  class Base {
4      virtual void f() { }
5  };
6  class Derivada: public Base { };
7  int main() {
8      try {
9          Base* a = new Base;
10         Base* b = new Derivada;
11         if (typeid(a) == typeid(b)) {
12             cout << "a is: " << typeid(a).name() << endl;
13             cout << "b is: " << typeid(b).name() << endl;
14             cout << "*a is: " << typeid(*a).name() << endl;
15             cout << "*b is: " << typeid(*b).name() << endl;
16         }
17     } catch (exception& e) {
18         cout << "Exception: " << e.what() << endl;
19     }
```

Definição de Macros (#define)

Usado para definir uma macro de Preprocessamento.

C++

```
1  #define getmax(a,b) ((a)>(b)?(a):(b))
2  #define X 5
3  #define Y 2
4
5  int main() {
6      int y;
7      y = getmax(X,Y);
8      cout << y << endl;
9      cout << getmax(7,X) << endl;
10     return 0;
11 }
```

Definição de Macros (#undef)

Usado para deletar uma macro definida.

C++

```
1  #define TABLE_SIZE 10
2  int table1[TABLE_SIZE];
3
4  #undef TABLE_SIZE
5
6  #define TABLE_SIZE 20
7
8  int table2[TABLE_SIZE];
```

Inclusões Condicionais

Permite incluir ou deletar certas partes de um programa dada uma condição.

C++

```
1  #if TABLE_SIZE>200
2  #undef TABLE_SIZE
3  #define TABLE_SIZE 200
4
5  #elif TABLE_SIZE<50
6  #undef TABLE_SIZE
7  #define TABLE_SIZE 50
8
9  #else
10 #undef TABLE_SIZE
11 #define TABLE_SIZE 100
12 #endif
13
14 int table[TABLE_SIZE];
```

References


Juan Soulié (2007)
The C++ Tutorial (www.cplusplus.com)


Kris Jamsa, Ph.D / Lars Klander (1999)
Programando em C/C++, A Bíblia

Fim.