Tecnologia em Análise e Desenvolvimento de Sistemas

2 Algoritmos e Estruturas de Dados

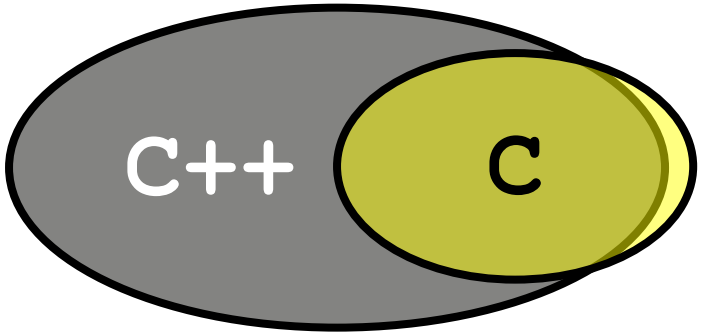
C++: Noções Básicas

Prof. Ramide Dantas

Comparativo entre C e C++

Foi criada para ser C++ é uma extensão de C

Mas não é 100% compatível com C



Comparativo entre C e C++

C	C++
Estruturada	Orientada a Objetos
`malloc` e **`calloc`**	**`new`**
`free`	**`delete`**
Passagem por valor	Passagem por referência
`stdio`	**`iostream`**
Variáveis declarada no início de um bloco	Variáveis declaradas em **qualquer** parte do bloco

Comparativo entre C e C++

C	C++
Inteiro como valor booleano	Tipo `bool`
Duas funções não podem ter o mesmo nome	Duas funções não podem ter o mesmo protótipo
Argumentos são sempre necessários	Valor defaut para os argumentos
Casts simples	Novos tipos de cast
`string` como array de caracteres	Tipo string

Retângulo em C

```
struct Retangulo {
        int altura;
        int largura;
};

int area(struct Retangulo *r) {
        return r->altura * r->largura;
}

int perimetro(struct Retangulo *r) {
                return 2 *(r->largura + r->altura);
}
```

Classes

São definições a partir das quais os objetos podem ser criados

As classes determinam quais são os atributos e métodos de um objeto

Sintaxe:

```
class nomeDaClasse {

    corpoDaClasse;

} listaDeObjetos;
```

Ponto e vírgula obrigatório.
Causa frequente de erros de compilação.

Retângulo em C++

```
class Retangulo {
      int largura;
      int altura;

      int area() {
            return largura * altura;
      }

      int perimetro() {
            return 2 *(largura + altura);
      }
} ret1;
// declara uma variável do tipo Retangulo
```

Visibilidade dos Membros

Membros de uma classe podem ser:

`public`

Podem ser acessados em qualquer lugar

`private`

Só podem ser acessados pelos membros da própria classe

`protected`

Podem ser acessados apenas por membros da própria classe ou das suas sub-classes

Exemplo

```
class Retangulo {
        int largura;
    private:
        int altura;
    public:
        int area() {
            return
                largura * altura;
        }
    protected:
        int perimetro() {
            return 2 *
                (largura + altura);
        }
};
```

Obs.: Por default todo membro de uma classe é considerado `private`

```
// Exemplo de acesso:
int main() {
    Retangulo r;
    // Errado:
    r.altura = 10;
    // Errado:
    r.largura = 40;
    // OK:
    int a = r.area();
    // Errado:
    a = r.perimetro();
}
```

Classes

Quando implementamos um método **dentro de uma classe** o compilador copia e cola o código toda vez que o método é chamado!

- O método é dito `inline`
- Isto torna o executável mais rápido
- Mas deixa o executável bem maior
- Só é bom para métodos muito curtos

Qual a solução?

- Utilizar o operador `::`

O Operador `::`

Permite a implementação de métodos fora da declaração da classe

- A classe passa a possuir apenas o protótipo do método
- O corpo pode ficar no mesmo arquivo ou em outro

Sintaxe:

nomeDaClasse::nomeDoMembro

Também podemos usar o modificador `inline` para que mesmo assim o método seja inline

Exemplo

```
class Retangulo {
    private:
        int largura;
        int altura;
    public:
        int area();
        int perimetro();
};

int inline Retangulo::area() {
    return largura * altura;
}
int Retangulo::perimetro() {
       return 2 *(largura + altura);
}
```

Apenas a declaração, sem corpo

Forçando a ser `inline`

Método não `inline`

Construtor

É um método especial que é chamado quando criamos um novo objeto

Deve possuir o mesmo nome da classe

Não possui retorno

É utilizado para inicializar os atributos da classe

Destrutor

Método especial que é chamado automaticamente quando um objeto está prestes a ser apagado da memória

Deve ter o mesmo nome da classe mas precedido por um ~

Assim como o construtor ele não possui retorno

Além disso, ele não pode ter parâmetros

Exemplo

```
class Retangulo {
    private:
        int largura;
        int altura;
    public:
        Retangulo(int a, int l);
        ~Retangulo() { } // destrutor padrão
};

Retangulo::Retangulo(int a, int l) {
      altura = a;
      largura = l;
}
```

Métodos get e set

get

Dá acesso aos atributos encapsulados de uma classe

```
int getLargura() {
    return largura;
}
```

set

Permite a modificação dos atributos encapsulados:

```
void setLargura(int l) {
    largura = l;
}
```

Exemplo

```
class Retangulo {
    private:
        int largura;
        int altura;
    public:
        int getAltura() { return altura;}
    protected:
        void setAltura(int a) {
            // evita um valor inválido
            if (a > 0) altura = a;
        }
};
```

Alocação de Memória

new

Aloca memória para um objeto

Retorna um ponteiro para a posição alocada

Chama o construtor do classe, se houver

```
Retangulo *r = new Retangulo(10, 15);
Retangulo *array = new Retangulo[10];
```

Desalocação de Memória

delete

Libera região de memória alocada previamente

Chama o destrutor da classe antes, se houver

```
delete r;
delete[] array;
```

Erros Comuns

```
// ...


Retangulo r = new Retangulo(10, 15);

// Errado: r não é um ponteiro!!!


delete r;

// Errado: r não é um ponteiro!!!


// ...
```

Classe e Estruturas

C++ permite a criação de estruturas **com métodos**

Estas são praticamente idênticas às classes

Porém todos os seu membros são `public` por default

Exemplo

```
struct Retangulo {

    int getAltura() {// public por default
        return altura;
    }

    void setAltura(int a) { // public por default
        if (a > 0) altura = a;
    }

    private:
        int largura;
        int altura;
};
```

E/S Padrão

C++ possui uma biblioteca de E/S chamada **`iostream`**

Esta possui alguns dispositivos predefinidos:

Nome	Tipo	Buffered	Descrição
`cin`	**`istream`**	Sim	Entrada padrão (normalmente o **teclado**)
`cout`	**`ostream`**	Sim	Saída Padrão (normalmente o **monitor**)
`clog`	**`ostream`**	Sim	Saída de erro padrão (normalmente o monitor)
`cerr`	**`ostream`**	Não	Saída de erro padrão (normalmente o monitor)

O tipo ostream

É um tipo de saída de dados

Podemos enviar dados para objetos deste tipo através do operador **<<** (operador de inserção)

O operador **<<** pode ser utilizado mais de uma vez na mesma sentença e não adiciona um **'\n'** no final da linha

```
cout << "IFPE\n";

//endl = a quebra de linha

cerr << "CIn" << endl;

clog << "C++" << "\n";
```

O tipo istream

É um tipo de entrada de dados

Podemos ler dados de objetos deste tipo através do operador **>>**

O operador **>>** também pode ser utilizado mais de uma vez na mesma sentença

A leitura só é realizada até um **'\r', '\n'** ou **' '**

```
int a;

float b;

cin >> a >> b;
```

Referências

```
void swap(int &a, int &b) {
    int temp = a;
    a = b;
    b = temp;
}
int main() {
    int a = 10;
    int b = 20;

    cout << "Antes:  a =  " << a << ", b = " << b << endl;
    swap(a, b);
    cout << "Depois: a =  " << a << ", b = " << b << endl;
    return 0;
}
```

Referências são "amarradas" a um objeto/variável apenas uma vez, na chamada de função ou inicialização.

Modularizando o seu programa

Quando queremos criar um projeto com diversas classes fazemos uso de algumas convenções:

Criamos um arquivo “.h” só com a definição da classe e os métodos inline

E um arquivo “.cpp” só com a implementação dos seus métodos

Retangulo.h

```
#ifndef _RETANGULO_H_
#define _RETANGULO_H_

class Retangulo {
        private:
                int altura;
                int largura;
        protected:
                void setAltura(int a);
                void setLargura(int l);
        public:
                Retangulo(int a = 1, int l = 1);
                int getAltura() { return this->altura; }
                int getLargura() { return this->largura; }
                int perimetro();
                int area();
                bool equals(Retangulo &r);
};

#endif //_RETANGULO_H_
```

Retangulo.cpp

```
#include "Retangulo.h"

Retangulo::Retangulo(int a, int l) {
	this->altura = a;
	this->largura = l;
}

void Retangulo::setAltura(int a) {
	this->altura = a;
}

void Retangulo::setLargura(int l) {
	this->largura = l;
}

...
```

Retangulo.cpp

```
...

int Retangulo::area() {
      return this->altura * this->largura;
}

int Retangulo::perimetro() {
      return 2 * (this->altura + this->largura);
}

bool Retangulo::equals(Retangulo &r) {
      return (this->altura == r.altura &&
              this->largura == r.largura);
}
```

Main.cpp

```
#include <iostream>
#include "Retangulo.h"

using namespace std;

int main() {
	Retangulo *r = new Retangulo(10, 15);
	Retangulo *arr = new Retangulo[10];
	Retangulo ret(10, 15);

	cout << "Area: " << r->area() << endl;
	cout << "Perimetro: " << r->perimetro() << endl;
	cout << "r = arr[0]: " << r->equals(arr[0]) << endl;
	cout << "r = ret: " << r->equals(ret) << endl;

	delete r;
	delete[] ar;

	return 0;
}
```

Referências

Adaptado do material em:

https://allanlima.wordpress.com/category/curso-de-c-para-programadores-java/

Tecnologia em Análise e Desenvolvimento de Sistemas

2 Algoritmos e Estruturas de Dados

C++: Noções Básicas

Prof. Ramide Dantas