A01 Revisão: conceitos básicos C++

📕 Ambientes de nomes

Os ambientes de nomes, ou *namespaces*, são usados para agrupar declarações relacionadas e evitar a colisão de nomes[1]. Por exemplo:

```
int x;
void inc(int i) { x += i; }

namespace Numero {
    int x;
    void inc(int i) {
         x += i;
     }
}

int main() {
    x = 100;   // esta é a variável global ::x
    inc(10);   // esta é a função global ::inc()
    ::inc(10); // esta é a função global ::inc()

    Numero::x = 200; // esta é a variável x de Numero
    Numero::inc(20); // esta é a função inc de Numero
}
```

Todos os nomes especificados em um namespace podem ser tornados acessíveis por meio de uma única diretiva de *namespaces*[1]:

```
using namespace Numero;
```

A conveniência do `using` é obtida ao custo de potenciais colisões de nomes. No geral, as diretivas `using` devem ser evitadas nos arquivos cabeçalho. Também é possível tornar disponível apenas um nome de um *namespace*[1]:

```
using Numero::inc;
inc(10);
```

📙 Comandos de entrada e saída

```
#include <iostream>
#include <iomanip>

using namespace std;

// Leitura da entrada padrão
int x = 0;
int y = 0;

cout << "Digite dois números: ";
cin >> x >> y;

// Escrita na saída padrão
cout << "x = " << x << endl;
cout << "y = " << y << endl;
```

📗 Formatação de dados

A biblioteca `iomanip` oferece um conjunto de funções para a formatação de dados na saída padrão. Todas as funções estão no namespace `std`.

- `setw`: define o número mínimo de caracteres da próxima saída.
- `setfill`: define o caractere que deverá ser usado para preencher os espaços.
- `right`: alinha os caracteres à direita.
- `left`: alinha os caracteres à esquerda.
- `setprecision`: define o número máximo de dígitos a serem exibidos.
- `fixed`: define o número de casas decimais para números em ponto

flutuante.

- `scientific`: define o número de casas decimais para números em ponto flutuante.

```
// setw: só modifica a próxima saída
cout << setw(20) << "uva" << "laranja" << "banana" << endl;

// setfill
cout << "Valor" << setfill('.') << setw(20) << ": " << 10 <<
endl;

// right / left
cout << "|" << right << setw(20) << "teste" << "|" << endl;
cout << "|" << left  << setw(20) << "teste" << "|" << endl;

// setprecision: define o número de dígitos
cout << setprecision(3) << 2.718283423 << endl; // 2.71

// fixed / scientific
cout.precision(4);  // Define a precisão dos números

cout << fixed      << 2.718283423 << endl; // 2.7183
cout << scientific << 2.718283423 << endl; // 2.7183e00
```

📘 Funções

Uma função é um trecho de código com um nome que recebe um conjunto de variáveis como parâmetros, e pode retornar um valor. Se a função é apenas declarada, ela termina com um ponto e vírgula. Se a função está implementada, o corpo de uma função deve ser um bloco[1].

```
// Declaração de um função
char funcao(char* str, int i);

// Implementação de uma função
char funcao(char* str, int i) {
```

```
    return str[i];
}
```

A chamada de uma função é feita através do nome da função seguida da lista de parâmetros.

```
char *frase = "Programar eh divertido!"

// Chamada da função
char a = funcao(1, 2);     // erro: primeiro parâmetro deve ser
char*
char b = funcao(frase);    // erro: número de parâmetros
char c = funcao(frase, 2); // ok
```

📕 Tipos de dados definidos pelo usuário

A linguagem C++ possui alguns tipos de dados pré-definidos (`bool`, `char`, `int`, `float`, `double`). Estes tipos são conhecidos pelo compilador que conhece quais as operações pode realizar com eles. Por exemplo, o compilador sabe quais operadores por utilizar com um tipo `int` e como as operações devem ser realizadas[1].

Além dos tipos pré-definidos, existem os Tipos de Dados Abstratos (TADs) que são criados pelos usuário. Estes tipos podem ser parte da biblioteca padrão, como `string`, `vector` e `ostream`, ou tipos criados pelos próprios programadores. As vantagens de criar tipos de dados próprios são[1]:

- **Representação**: um tipo conhece o modelo dos dados o qual está representando.
- **Operações**: um tipo conhece as operações que podem ser aplicados no modelo de dados.

Um modelo de dados pode ser representado através de uma `struct`.

```
struct Data {
    int dia;
    int mes;
    int ano;
};
```

Na `struct` todas as variáveis são visíveis por padrão. As `structs` são usadas principalmente para criar estruturas de dados nas quais os membros podem ter qualquer valor, *i.e.*, não podem ser definidas invariantes que façam sentido.

📙 Criação de módulos

Um módulo é usado para definir e organizar os TADs. A convenção é criar um arquivo cabeçalho `.h` e um arquivo fonte `.cpp` com o nome do TAD. No arquivo cabeçalho são definidos[1]:

- Inclusão de bibliotecas
- Definição de constantes / macros
- Definição de tipos enumerados
- Definição de apelidos
- Definição de *namespaces*
- Representação do modelo de dados
- Operações para manipular o modelo de dados

Arquivo cabeçalho

O arquivo cabeçalho deve incluir o bloco `#ifndef` para garantir que o arquivo não seja incluído mais de uma vez durante o processo de compilação. Alguns compiladores aceitam a diretiva `#pragma once` que tem o mesmo efeito.

```
#ifndef __MODULO_H__
#define __MODULO_H__

struct Data {
    int dia;
    int mes;
    int ano;
};

bool dataVerificarAnoBissexto(Data &data);
void dataImprimir(Data &data);

#endif
```

> ⚠️ **Atenção**
>
> - **Variáveis** não devem ser declaradas no arquivo cabeçalho.
> - **Funções** não devem ser implementadas no arquivo cabeçalho.

Arquivo fonte

O arquivo fonte deve incluir a implementação das operações definidas no arquivo cabeçalho. Por padrão, o arquivo fonte deve sempre incluir o arquivo cabeçalho que define o módulo[1].

```
#include "data.h"

bool dataVerificarAnoBissexto(Data &data) {
// ...
}

void dataImprimir(Data &data) {
// ...
}
```

👨🏼‍💻 Atividade prática

1. Escreva um módulo que represente o modelo de dados de um relógio. O relógio deve conter os atributos de hora e minuto. As seguintes operações devem ser definidas:
   - Inicialização do relógio
   - Definir um horário (validar se a hora é valida)
   - Imprimir os dados do relógio (imprimir no formato `hh:mm`)
2. Escreva um programa que crie 3 relógios e inicialize com os valores fornecidos pelo usuário.
3. Escreva um programa que leia os dados de *n* produtos informados pelo usuário e imprima os dados em formato tabular:

```
| nome       | preco       | quantidade |
|------------|-------------|------------|
| camiseta   | R$    70.00 |          2 |
| bermuda    | R$    59.00 |          1 |
| bone       | R$     9.00 |          1 |
```

📝 Atividade teórica

1. Qual o objetivo do uso de namespaces?
2. Como resolver o problema de conflito caso já exista um namespace com mesmo nome no projeto?
3. Quais são as outras maneiras de criar uma `struct` herdadas do C?
4. Descreva como seria a organização de um módulo para representação de uma estrutura de dados do tipo Pilha.
   a. Qual seria o nome dos namespace, arquivos cabeçalho e fonte?
   b. Como seria o modelo de dados?
   c. Quais seriam as operações a serem realizadas?

📚 Referências bibliográficas

[1] STROUSTRUP, Bjarne; LISBÔA, Maria Lúcia Blanck; LISBÔA, Carlos Arthur Lang (Trad.). **A linguagem de programação C++**. 3 ed. Porto Alegre, RS: Bookman, 2000. ↩