Escola de Engenharia
Universidade do Minho

Complementos de Programação de Computadores – Aula Teórica 1b

Introdução ao C++

Luís Paulo Reis

lpreis@dsi.uminho.pt

Professor Associado do Departamento de Sistemas de Informação, Escola de Engenharia, Universidade do Minho, Portugal

(Slides Baseados em L.P.Reis et al., 2006 e Paulo Cortez, 2011)

Escola de Engenharia
Universidade do Minho

Breve Historial

1972 Primeira versão da linguagem **C** criada nos laboratórios Bell (da AT&T) por Dennis Ritchie, e implementada num computador DEC PDP-11, baseada nas linguagens B e BCPL

1978 **C clássico** descrito no livro "The C Programming Language", por **Brian Kernighan** e **Dennis Ritchie**, dos laboratórios Bell

1983 Primeira versão da linguagem **C++** (pelo menos com esse nome), uma extensão ao C (sobretudo com facilidades de programação orientada por objectos) criada por **Bjarne Stroustrup** nos laboratórios Bell (da AT&T)

1988 **C standard** (ANSI C) descrito na segunda edição do livro de Kernighan e Ritchie e aprovado pelo comité ANSI (aprovado pelo comité ISO em 1990)

1995 A linguagem de programação **Java** é criada na Sun Microsystems, baseada em C e C++ e incorporando características doutras linguagens orientadas por objectos (em geral é mais "limpa" mas menos eficiente que C++)

1997 **C++ standard** descrito na 3ª edição do livro "The C++ Programming Language" de Bjarne Stroustup e aprovado pelo comité ISO em 1998

2001 Criada a linguagem C# na Microsoft , baseada em Java e C++

Filosofia

Em "In The Design and Evolution of C++ (1994)", Bjarne Stroustrup descreve algumas regras que ele utiliza para desenvolver o C++:

- Desenvolvido para ser uma linguagem tipada estaticamente e de proposta geral, tão eficiente e portável como o C
- Suporte para múltiplos paradigmas
- Fornecer ao programador escolhas, mesmo que seja possível ao programador escolher a opção errada
- O mais compatível com C possível, fornecendo transições simples para código C
- Evita fornecer facilidades que são específicas a certas plataformas ou a certos grupos de programadores
- Não exige overhead para facilidades que não são utilizadas
- Poder ser utilizado mesmo sem um ambiente de desenvolvimento sofisticado

Vantagens

- Programas novos são desenvolvidos em menos tempo, devido à reutilização de código
- É mais fácil criar e manipular novos tipos de dados
- A gestão de memória é mais simples e transparente
- Os programas conterão menos erros, devido à maior exigência da verificação de erros por parte do compilador
- Facilita o encapsulamento de dados
- Permite a programação orientada por objectos: classes e objectos, encapsulamento, herança, polimorfismo, …)

O C++ (bem como a programação Orientada por Objectos - OO) não é uma solução universal para todos os problemas de programação!

Novas Potencialidades do C++

- Declarações como instruções
- Tipagens Function-like
- New/delete
- Novos Tipos: bool e string
- Tipos Referência
- Funções Inline
- Argumentos por Defeito
- Overload de Funções
- Namespaces
- Classes (herança, funções membro, funções virtuais, classes abstratas, e construtores)
- Overloading de Operadores
- Templates
- Operador ::
- Exception handling
- Identificação em Runtime de tipos
- Verificação de tipos (type checking) mais detalhada do que o C
- Comentários começando com duas barras ("//") foram reintroduzidos

Compiladores C++

- **GNU gcc ou g++: open source, permite compilar em Unix, Linux, Mac OS, ...**
- **MinGW: versão do GNU gcc para Windows**
- **Dev C++ com MinGW: Windows, aulas práticas**
- **MS Visual Studio: Windows, aulas práticas,...**
- **Turbo C++: windows**
- **Borland C++: windows**

Ver mais em: http://en.wikipedia.org/wiki/List_of_compilers

Primeiro programa em C++

Directiva para incluir *header file* (`.h`) da biblioteca *standard* `iostream` (*streams* de entrada e saída de dados)

Função principal de um programa em C++

Standard output stream (normalmente o ecrã)

Envia o dado da direita para o *stream* da esquerda

Segue-se a convenção habitual de a função `main` retornar 0 em caso de sucesso

Endl equivalente a printf"\n"

```
#include <iostream>

main()
{
   cout << "hello, world"<< endl;
   return 0;
}
```

helloWorld.C (Unix) ou
helloWorld.cpp (Windows)

No ecrã aparece:

```
hello, world
```


Segundo programa em C++

```
// calcula o máximo e a média de um conjunto de valores reais
#include <iostream> // para usar "cout" e "cin"

main()
{
    cout << "Quantos valores são? ";
    int n;
    cin >> n;
    float soma = 0.0, maximo;
    for (int i = 1; i <= n; i++)
    {
        cout << "x" << i << "? ";
        float x;
        cin >> x;
        soma += x; // mesmo que soma = soma + x
        if (i == 1 || x > maximo)
            maximo = x;
    }
    cout << "máximo=" << maximo << "\n";
    cout << "média=" << soma / n << "\n";
    return 0;
}
```

```
Quantos valores são? 4
x1? 18
x2? 15.5
x3? 14.5
x4? 17
máximo=18
média=16.25
```

Um programa em C++

Duas formas de definir um programa em C++

```
int main()
{
    ...
}
```

```
int main(int argc, char *argv)
{
    ...
}
```

– Instrução `return` é opcional : `return 0` é implícito no final

Primeiras extensões ao C

- **Nos exemplos anteriores notam-se as seguintes extensões:**
  - declaração de variáveis em qualquer ponto de um bloco (a variável existe até terminar a execução do bloco)
  - declaração de variáveis na parte de inicialização do ciclo `for` (a variável existe até terminar a execução do ciclo)
  - comentários começados com `//` (terminam no fim da linha)
  - entrada e saída de dados mais segura e simples com *streams*

- **Notar, no entanto, que o C é um subconjunto do C++, pelo que todas as *features* do C continuam disponíveis**

Namespaces e Operador Escopo::

Namespaces:

- Permitem definir símbolos num contexto alargado, chamados de namespace, sendo utilizados para evitar conflitos entre funções
- Para já importa saber que a maior parte dos compiladores exige a seguinte linha de código, a seguir aos includes:

```
using namespace std;
```

Operador de Escopo :: (utilizado para distinguir variáveis locais de globais):

```
#include <cstdio>
int conta = 50; //variável global
int main() {
    for(int conta = 1; conta <10; conta++) { // local
          printf("\%d %d\n", ::conta // global
          , conta); // local
    }
    return (0);
}
```


Variáveis Locais

- Em C, as variáveis locais tem de ser definidas no início de cada função. No entanto, em C++, as variáveis podem ser definidas em qualquer local!

```
int main() {
    for(int i=0;i<10;i++)
    for(int j=0;j<i;j++);
}
```

- Atenção: neste caso, as variáveis i e j só são válidas dentro dos ciclos!
- Regra: variáveis locais devem ser definidas no início das funções, após o primeiro { ou imediatamente antes de serem necessárias, mas com o devido cuidado

Sobreposição/overloading de funções

- **Sobreposição/overloading de funções:**
  - É possível definir funções diferentes com os mesmos nomes
  - Nesse caso, as funções distinguem-se pelos argumentos de entrada

Exemplo:

```
# include <cstdio>
void show(int val){ printf("%d\n", val);}
void show(double val){ printf("%lf\n", val);}
void show(char *val){ printf("%s\n", val);}
int main()
{
  show(3.1415);
  show("bom dia");
  show(4);
}
```


Argumentos Implícitos

- **É possível sugerir valores implícitos para argumentos de funções, que são activados quando o utilizador não os define:**

```
void show(char *str="Bom dia!\n") {   printf("%s",str);  }
int main() {
  show("Texto explicito!!!\n");
  show(); // igual a show("Bom dia!\n");
}
```

- **Os argumentos implícitos devem ser declarados nos protótipos das funções (ficheiros .h ou .hpp)**
- **É assumida uma leitura a partir da direita para a omissão**

```
void dois(int a = 1, int b = 4)  { printf("%d %d\n", a, b); }
int main() {
   dois();   dois(20);   dois(20,10);
}
```

Outras Extensões

- **Conversão Estática**
  - Serve para converter um tipo de dados em outro tipo de dados
  - Uso: static_cast<tipo>(exp) ;
  - Exemplos:

```
int a = static_cast<int>(12.45);
float media = static_cast<float>(45/27);
char c = static_cast<char>(a);
```

- **Constantes:**
  - A palavra const define uma constante (não pode ser alterada);
  - Embora também se utilize em C, em C++ é muito mais utilizada;
  - Exemplos:

```
int const a=5;
double const pi=3.1415;
void imprime(char const *s);
```


Entrada e saída de dados com streams

- **`cout << exp1 << exp2 << ...`**
  - escreve (insere) no *stream* de saída os valores das expressões indicadas
  - **`cout`** é o *standard output stream* (normalmente conectado ao ecrã)
  - "**<<**" está definido para tipos de dados *built-in* e pode ser definido para tipos de dados definidos pelo utilizador
- **`cout << endl`**
  - escreve caracter de mudança de linha e despeja o *buffer* de saída
- **`cout.put(c)`**
  - escreve um caracter no *stream* de saída
- **`cin >> var1 >> var2 >> ...`**
  - lê (extrai) do *stream* de entrada valores para as variáveis da direita
  - **`cin`** é o *standard input stream* (normalmente conectado ao teclado)
  - ">>" está definido para tipos de dados *built-in* e pode ser definido para tipos de dados definidos pelo utilizador
  - salta caracteres "brancos" (espaço, *tab*, *newline, carriage return, vertical tab* e *formfeed*), que servem para separar os valores de entrada

Entrada e saída de dados com streams

- operador ">>" devolve falso se falhar leitura

```
if (!(cin >> x)) cerr << “Erro na leitura de x \n”;
```

- **`cin.eof()`**
  - testa se chegou ao fim do *stream* de entrada
- **`cin.get()`**
  - lê um caracter do *stream* de entrada; não salta caracteres brancos; retorna EOF se encontrar o fim do *stream* (no teclado é normalmente indicado com ctrl-Z em Windows e ctrl-D em Unix)
- **`cerr`** – *standard error*
  - para escrever mensagens de erro
- **`cin`** é uma variável do tipo *istream* (input *stream*) definidas em *"iostream.h"*
- **`cout`**, **`cerr`** são variáveis do tipo `ostream` (*output stream*) definida em "iostream.h"

Manipulação de ficheiros com streams

```
// Programa que copia o conteúdo do ficheiro f1 para o ficheiro f2
#include <fstream>
main()
{
  ifstream  origem ("f1");  // define variável e abre ficheiro
                            // para leitura
   if (!origem)
    { cerr << "Erro a abrir ficheiro f1\n"; return -1; }

  ofstream  destino ("f2"); // idem, para escrita
  if (!destino)
    { cerr << "Erro a abrir ficheiro f2\n"; return -1; }

  char c;

  while ( (c = origem.get()) != EOF )
       destino.put(c);

  if (!origem.eof() || !destino /*em bool dá false após erro*/)
    { cerr << "Erro\n"; return -1; }
  return 0;
} // ficheiros são fechados automaticamente
```

Manipulação de Ficheiros com *streams*

- **`ifstream origem ("f1.txt");`**
  - define variável e abre ficheiro para leitura
- **`ofstream destino ("f2.txt");`**
  - define variável e abre ficheiro para escrita
- **`f.close()`**
  - fecha o ficheiro
- **`f.open(nome)`**
  - abre o ficheiro com o nome indicado
- **`!origem`**
  - testa se conseguiu abrir ficheiro
- **`!destino`**
  - testa se conseguiu criar ficheiro
- **`origem.get()`**
  - lê informação (carácter) do ficheiro
- **`destino.put(c)`**
  - escreve informação (carácter) no ficheiro
- **`!origem.eof()`**
  - testa se chegou ao fim do ficheiro

O tipo string

```
#include <string>    // definições no espaço de nomes std
#include <iostream>  // idem

using namespace std; // traz nomes de std para global

main()
{
string primeiro, ultimo, completo;
    cin >> primeiro; // lê (pára em espaço, tab ou newline)
    cin >> ultimo;   // idem
    if (primeiro == ultimo)  // compara
       cout << "Primeiro e último nome iguais!\n";
    completo = primeiro + " " + ultimo; // concatena e copia
    cout << completo; // escreve
    return 0;
}
```

Mais prático que usar arrays de caracteres!

O tipo string (cont.)

- **Inicialização:**

```
string mes = "Janeiro";
string mes("Janeiro");
```

- **`s.length()`** retorna o comprimento da string `s`
  - chama o membro-função `length` no objecto `s`
- **`s[i]`** refere-se ao caracter que se encontra na posição `i` da string `s` (`0 <= i <= s.length()-1`)
- Comparação de strings faz-se com operadores de comparação habituais (>, >=, <, <=, ==, !=)
- **`getline(cin,s)`**
  - lê uma linha do *`input`* para a string `s`
  - o primeiro argumento também pode ser do tipo `ifstream` (ficheiro)

O tipo string (cont.)

- **`s1 += s2`**
  - concatena `s2` no fim de `s1`
  - `s2` pode ser do tipo `string`, `char *` ou `char`
- **`s.substr(i,n)`**
  - devolve substring de tamanho `n` e início na posição `i` de `s`
  - segundo argumento pode ser omitido (nesse caso é até ao fim da string)
- **`s1.find(s2)`**
  - devolve a posição inicial da primeira ocorrência de `s2` em `s1` ou `string::npos` se não existir nenhuma ocorrência de `s2` em `s1`
  - `s2` pode ser do tipo `string` ou `char *`
- **`s.c_str()`**
  - dá a string em C (tipo `char *`)
- **E muito mais!**

O tipo bool

- Em C, não existem booleanos, existe apenas a convenção
  - 0 é falso
  - ≠0 é verdadeiro
- Em C++ existe o tipo `bool`
- As constantes deste tipo são `true` e `false`
- Booleanos são convertíveis implicitamente para inteiros seguindo as convenções do C
- Operadores lógicos e de comparação dão resultado do tipo `bool`

Passagem de argumentos por referência

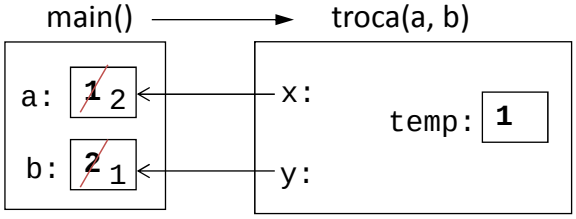
```
// Troca valores de variáveis passadas na chamada
void troca(int & x, int & y)
{
    int temp = x;
    x = y;
    y = temp;
}

main() // Testa a função anterior
{
        int a = 1, b = 2;
        troca(a, b); // troca valores de a e b
        cout << "a =" << a << '\n';
        cout << "b =" << b << '\n';
}
```

Uma variável do tipo **T &** (referência para **T**) é uma referência (*alias* ou nome alternativo) para um objecto do tipo **T**

a=2
b=1

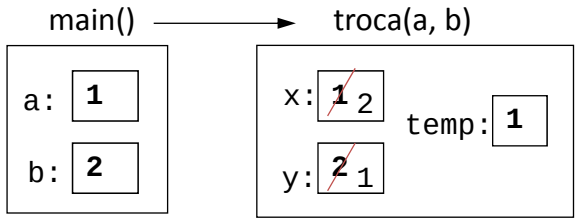
Mais simples do que trabalhar com apontadores!

Usar quando se pretende que a função chamada altere valores de variáveis passadas na chamada



Passagem de argumentos por valor

```
// Função inútil!!
void troca(int x, int y)
{
    int temp = x;
    x = y;
    y = temp;
}

main() // Testa a função anterior
{
        int a = 1, b = 2;
        troca(a, b); // ???
        cout << "a =" << a << '\n';
        cout << "b =" << b << '\n';
}
```

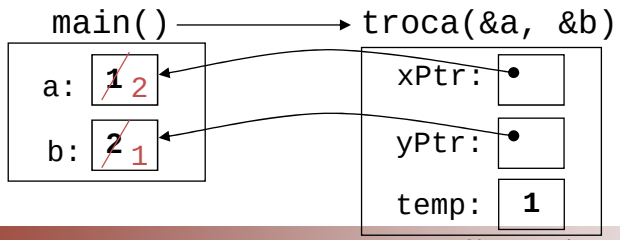


a=1
b=2

Passagem por referência com apontadores

```
// Função troca com apontadores em vez de referências
void troca(int *xPtr, int *yPtr)
{
    int temp = *xPtr;
    *xPtr = *yPtr;
    *yPtr = temp;
}

main() // Testa a função anterior
{
        int a = 1, b = 2;
        troca(&a, &b); // troca valores de a e b
        cout << "a =" << a << '\n';
        cout << "b =" << b << '\n';
}
```

- Passa o endereço da variável
- Permite que a função chamada altere o valor (conteúdo) da variável
- **Pouco útil**, porque é mais simples usar referências!



Mais complicado do que trabalhar com referências!

Passagem de argumentos

```
// passagem por valor – troca não funciona
void trocaNaoFunc(int x, int y)
{
    int temp = x;
    x = y;
    y = temp;
}
// C – uso de apontadores
void trocaAp(int *x, int *y)
{
    int temp = *x;
    *x = *y;
    *y = temp;
}
// C++ – uso de referências
void trocaRefp(int &x, int &y)

    int temp =´x;
    x = y;
    y = temp;
}
```


Passagem de argumentos

```
int main()
{
    int a = 4 , b = 6;
    trocaNaoFunc(a,b);
    cout << “a = “ << a << ‘\n’;
    cout << “b = “ << b << ‘\n’;
    trocaAp(&a,&b);
    cout << “a = “ << a << ‘\n’;
    cout << “b = “ << b << ‘\n’;
    trocaRef(a,b);
    cout << “a = “ << a << ‘\n’;
    cout << “b = “ << b << ‘\n’;


    return 0;
}
```

Escola de Engenharia
Universidade do Minho

Resumo das principais extensões ao C

- Comentários começados em *// vão até ao fim da linha*
- Definição de variáveis a meio de um bloco e na parte de inicialização da instrução `for`
- Biblioteca alternativa de entrada e saída de dados baseada em *streams*, mais segura e mais simples
- Tipo `string` (da biblioteca standard) mais fácil de usar do que as strings *built-in* herdadas do C
- Tipo `bool` com valores `true` e `false`
- Passagem de arguments por referência (dispensa apontadores)
- Classes e Programação Orientada a Objectos! -> Próximas Aulas!

Escola de Engenharia
Universidade do Minho

Complementos de Programação de Computadores – Aula Teórica 1b

Introdução ao C++

Luís Paulo Reis

lpreis@dsi.uminho.pt

Professor Associado do Departamento de Sistemas de Informação, Escola de Engenharia, Universidade do Minho, Portugal

(Slides Baseados em L.P.Reis et al., 2006 e Paulo Cortez, 2011)