Introdução ao C para programadores de Java

Parte 1

O Java herdou do C uma grande parte da sua sintaxe, assim como tipos, instruções e operadores.

Sendo o Java uma linguagem de programação por objectos, os programas estruturam-se em classes, com os respectivos atributos e métodos. Por seu lado, os programas em C, uma linguagem imperativa, estruturam-se em funções. Se, no Java, as instruções do programa se encontram nos métodos, em C, todas as instruções estão dentro das funções. Se retirarmos de um programa em Java o invólucro das definições das classes, obtemos um programa que *parece* um programa escrito em C.

O resto desta 1ª parte apresenta, primeiro, coisas que se mantêm e, depois, algumas das diferenças entre o Java e o C.

Coisas que se mantêm

Mantêm-se as estruturas de controlo: `if`, `while`, `for` (com um *caveat*), `do/while` e `switch`. Também se mantêm as instruções `return`, `break` e `continue` (as duas últimas só sem argumento).

Mantém-se a sintaxe da instrução de afectação.

Mantém-se a maioria dos operadores. No entanto, o C não tem o operador `+` para concatenar *strings*, nem o operador `>>>`.

Mantém-se a sintaxe das *strings* (delimitadas por aspas) e dos caracteres (delimitados por plicas e limitados a 8 bits). Mantém-se também a representação especial de alguns caracteres, como `\n` (fim de linha), `\t` (tab), `\\` (*backslash*), `\"` (aspas) e `\'` (plica), assim como a notação `\octal`, onde *octal* é um número em base 8, entre 0_8 e 377_8.

Mantém-se a sintaxe dos comentários.

Mantém-se o uso do tipo `void` para as funções que não devolvem qualquer valor.

A sintaxe da definição de uma função em C é a sintaxe da definição de um método em Java, exceptuando a cláusula `throws`.

Tipos primitivos

Os principais tipos primitivos do C são os apresentados a seguir.

Valores inteiros

- `short` (inteiro de 16 bits)
- `int` (32 bits)
- `long` (32 ou 64 bits, dependendo da arquitectura da máquina)
- `long long` (64 bits)
- `char` (8 bits)

Os tamanhos indicados são os tamanhos comuns para os valores destes tipos. Todos estes tipos correspondem a inteiros com sinal, mas para todos existe uma variante `unsigned` (sem sinal), que permite representar só números não negativos. Por exemplo, o tipo `unsigned int` é a variante sem sinal do tipo `int`.

Em C não existe o tipo `byte`, podendo o tipo `char` ser usado em vez dele.

Valores reais

Tipos `float` (precisão simples, com 32 bits) e `double` (precisão dupla, com 64 bits).

Valores lógicos

Tipo `bool`, com valores `true` e `false`. (Ver abaixo o que fazer para poder usar este tipo.)

Em C, qualquer valor pode ser testado. O teste falha (é `false`) se o valor for zero (ie, se todos os seus bits tiverem o valor 0), e sucede (é `true`) para qualquer valor diferente de zero (ie, se algum bit tiver o valor 1).

Funções

Definição

A sintaxe das definições de funções tem a forma seguinte, semelhante à das definições dos métodos no Java:

```
tipo nome(argumentos...)
{
  instruções
}
```

O compilador de C precisa de ter informação sobre as funções usadas no código antes de elas serem usadas. Para isso acontecer, a definição de uma função deve aparecer no ficheiro *antes* do seu uso.

Quando uma função não devolve nenhum valor (ie, trata-se de um procedimento), o seu `tipo`será `void`.

No exemplo seguinte, o tipo `void` no lugar da declaração dos argumentos significa que a função não tem qualquer argumento:

```
unsigned int sem_argumentos(void)
{
  return 123456789;
}
```

Em C, não pode haver duas funções com o mesmo nome.

Uso de funções

Onde em Java pode aparecer a invocação de um método, nas formas *`Classe`*`.`*`método`*`(...)` ou*`objecto`*`.`*`método`*`(...)`, em C pode aparecer a invocação de uma função *`função`*`(...)`.

Função `main`

A função `main` de um programa em C desempenha o mesmo papel que o método `main` de um programa em Java.

Esta função devolve um valor de tipo `int` e uma instrução

`return` *`código`*`;`

tem, nesta função, o mesmo efeito que `System.exit(`*`código`*`);` em Java.
Um exemplo de função `main` é:

```
int main(void)
{
  return 0;
}
```

Nome dos ficheiros

Os ficheiros com código C devem ter extensão `.c`, por exemplo, `programa.c`.

A extensão `.h` é usada para ficheiros que só contêm declarações, ie, onde se definem tipos e nomes (de funções ou de variáveis). O *h* vem de *header*.

Inclusão de ficheiros

Quando são usadas funções definidas externamente, a directiva `#include` do pré-processador de C (correspondente à directiva `import` do Java) permite importar as suas declarações para o ficheiro onde as funções são usadas. Estas declarações fazem parte do conteúdo dos ficheiros `.h`.

A directiva `#include` tem duas formas:

```
#include <ficheiro.h>
#include "ficheiro.h"
```

A primeira forma é usada quando se pretendem importar ficheiros pertencentes ao sistema ou ao compilador. A segunda utiliza-se para importar ficheiros do programador.
Pela razão apontada acima, todas as directivas `#include` devem ocorrer no início do ficheiro em que aparecem.

O tipo `bool` é uma adição ao C relativamente recente e, para o usar e ter acesso às constantes`true` e `false`, o programa deve incluir o ficheiro `stdbool.h` através da directiva:

```
#include <stdbool.h>
```

Escrita na consola

A função `printf` permite a um programa em C escrever mensagens na consola. O seu uso tem a forma:

`printf(`*formato*`, `*expressão*$_1$`, `*expressão*$_2$`, ..., `*expressão*$_n$`);`

com $n \geq 0$.

O formato é uma *string* que controla como o valor de cada uma das expressões será escrito. O conteúdo do formato é escrito literalmente na consola, excepto quando aparece o carácter `%`. Quando este aparece, o que é escrito depende do que se lhe segue no formato. Algumas hipóteses são:

- `%` – é escrito o carácter `%`;
- `d` – é escrito o valor inteiro da próxima expressão a escrever;
- `c` – é escrito carácter correspondente ao valor da próxima expressão a escrever;
- `s` – é escrita a *string* correspondente à próxima expressão a escrever;
- `f` ou `g` – é escrito o valor real da próxima expressão a escrever.

(Há muitas outras possibilidades e variantes, descritas na documentação da função, acessível, em Linux, através do comando `man 3 printf`.)

O formato deverá conter tantas ocorrências de `%` (seguido de qualquer coisa diferente de `%`) quantas as expressões que o seguem.

A presença do carácter `'\n'` no fim do formato faz terminar a linha que aparece na consola (ou seja, o que quer que seja escrito a seguir, sê-lo-á na linha seguinte da consola).

Para usar esta função (ou outras relacionadas com escrita/leitura), o programa deve incluir o ficheiro `stdio.h`.

1. Se `idade` for uma variável inteira com valor 27 e `igual` for do tipo `char` e tiver o valor `'='`, a instrução:

   ```
   printf("idade + 10 %c %d\n", igual, idade + 10);
   ```

   escreverá na consola:

   ```
   idade + 10 = 37
   ```

2. Se `taxa` for uma variável real com valor 0.75, a instrução:

   ```
   printf("50%% de %f é %g\n", taxa, taxa / 2);
   ```

   escreverá na consola:

   ```
   50% de 0.750000 é 0.375
   ```

Leitura de inteiros

A leitura de um valor inteiro, introduzido através do teclado, pode ser feita recorrendo à função `scanf`, cujo uso é semelhante ao da função `printf`.

A chamada seguinte lê um valor para a variável inteira `idade`:

```
scanf("%d", &idade);
```

Repare no uso, *obrigatório*, de `&`, antes do nome da variável inteira para a qual se quer ler o valor.

(*Voltaremos a olhar para esta função...*)

Compilar código C

O `gcc` é um compilador de distribuição livre, e está habitualmente presente nos sistemas Linux (mas também está disponível para Windows).

O comando seguinte invoca o `gcc` para compilar o programa em C contido no ficheiro `programa.c` e diz-lhe para chamar `programa` ao executável criado:

```
gcc -Wall programa.c -o programa
```

O executável criado pode, depois, ser executado através do comando:

```
./programa
```

que indica que queremos executar o programa contido no ficheiro `programa` da directoria corrente (`.`).

(A opção `-Wall` indica ao compilador que deve assinalar todas as ocorrências, no código, de coisas que não constituem erros mas que podem não estar correctas.)

O `for` e o `gcc`

Algumas versões do `gcc` compilam código C de acordo com o *standard* C89, também conhecido como ANSI C, com algumas extensões. Este *standard* não contempla a possibilidade de declarar variáveis na zona de inicialização do ciclo `for`, que só apareceu no C99.

Se o `gcc` assinalar um erro numa instrução do tipo:

```
for (int i = 0; i < MAX; ++i) ...
```

isso significa que ele não está a compilar o código de acordo com o *standard* C99, e é necessário forçá-lo explicitamente a fazê-lo, incluindo no comando de compilação a opção `-std=c99` ou `-std=gnu99`, como no comando:

```
gcc -std=gnu99 -Wall programa.c -o programa
```

(Usando `gnu99`, em vez de `c99`, tem-se acesso a algumas funções que não fazem parte do C99 e que podem ser úteis, como a função strdup.)