Departamento de Matemática da Universidade de Coimbra

Métodos de Programação II	Teste Teórico-Prático 1	19/10/2007

Leia atentamente o enunciado das perguntas antes de iniciar a sua resolução.

Exercício 1

Explique, a sintaxe e o funcionamento de um ciclo **for** em linguagem C. Pode usar um PEQUENO exemplo se achar necessário.

Resolução:

O ciclo *for* na linguagem C apresenta-se com a seguinte sintaxe:

for(expressão 1; condição lógica; expressão 2)
intrução; (ou bloco de instruções)

onde:

- A *condição lógica* é a cabeça do ciclo, controlando a sua repetição: se avaliada para *Verdade* repete o ciclo; caso contrário, devolve o controlo da sequência de instruções para aquela que vem imediatamente a seguir ao fim do corpo do ciclo.
- A *expressão 1*, por vezes referida como inicialização, é constituída por uma ou mais instruções de atribuição, que podem, inclusive, usar chamadas a funções, quer locais, quer pré-definidas.

  É **efectuada uma única vez** e sempre antes do teste da cabeça do ciclo, pelo que, mesmo que o ciclo nunca se efectue (a primeira avaliação da condição lógica é *Falsa*), esta atribuição (ou este conjunto de atribuições) efectua-se sempre.
- A *expressão 2*, por vezes referida como actualização, obedece à mesma descrição de 1, no entanto, só é efectuada **após** a(s) instrução (intruções) do corpo do ciclo e **antes** da próxima avaliação da condição de teste do ciclo.

Qualquer das expressões e (ou) condição podem ser omitidas mas os **;** não!

Exercício 2

Preveja e explique a saída da implementação do seguinte código, supondo que são introduzidos os valores 0.5, 2.5, 0.5, 1.8, respectivamente:

```
typedef struct teste1{  double x, y; } Misterio;

void funcao1( Misterio A, Misterio *B)
{ Misterio C;
       B->x = A.x * B->x - A.y * B->y;
       B->y = A.x * B->y + A.y * B->x;
}

main()
{ Misterio X, Y;
       printf("Insira 4 valores reais\n");
       scanf("%g %g %g %g", &(X.x), &(X.y), &(Y.x), &(Y.y));
       funcao1(X, &Y)
       printf("%d %d %d %d\n", X.x, X.y, Y.x, Y.y);
       exit(0);
}
```

Resolução:

A estrutura Misterio pode perfeitamente representar números complexos em C, pelo que, nesse caso, o procedimento *funcao1* seria a implementação da multiplicação algébrica de números complexos. Assim, a instrução:

```
scanf("%g %g %g %g", &(X.x), &(X.y), &(Y.x), &(Y.y));
```

atribui os valores seguintes: $X.x = 0.5, X.y = 2.5, Y.x = 0.5$ e $Y.y = 1.8$. Como, na chamda da *funcao1*, o segundo parâmetro (Y) passa por referência, vem alterado de acordo com as operações aí efectuadas e, portanto, a instrução `printf` vai ter como resultado a escrita dos seguintes valores:

$$0.5 \quad 2.5 \quad -4.25 \quad -9.725$$

Note-se que, após a instrução:

```
B->x = A.x * B->x - A.y * B->y;
```

o valor de `B->x` (ou de `Y.x`, o que é o mesmo por ser passado por referência) vem alterado para -4.25. Logo, este é o valor que vai ser usado na segunda instrução:

```
B->y = A.x * B->y + A.y * B->x;
```

Exercício 3

Responda a **uma e sómente uma** das seguintes alíneas:

3.1) Descreva cada uma das funções para gestão de memória dinâmica da linguagem C, explicando a sua sintaxe e o seu funcionamento.

Resolução:

As funções para gestão dinâmica de memória da linguagem C são as seguintes:

- `void* malloc(unsigned size)`, função que devolve um ponteiro para uma zona reservada de memória de tamanho *size* ou `NULL`, caso não haja memória suficiente disponível;
- `void* calloc(unsigned num, unsigned size)`, que devolve um ponteiro para uma zona reservada de memória de tamanho *size*, inicializada de acordo com o tipo correspondente ao pedido através do tamanho, ou `NULL`, caso não haja memória suficiente disponível;
- `void* realloc(void *ptrant, unsigned size)`, função que devolve um ponteiro para uma zona reservada de memória de tamanho *size*, reorganizada (aumentadad ou diminuida) relativamente ao ponteiro `ptrant`, ou `NULL`, caso não haja memória suficiente disponível;
- `void free(void *ptr)` que devolve (liberta) memória reservada com uma das funções anteriores.

3.2) O seguinte programa contém alguns erros. Corrija esses erros, explicando o porquê de cada erro.

```
int i = 0, n, *ptrN;
char c, *ptrC;

c = '\0';
ptrC = c;
ptrN = (int) malloc(10*sizeof(int));
while(i < 10){ *(ptrN+i) = i; i++; }
printf("c = %c  ptrC -> %c   ptrN -> %d\n", c, *ptrC, *ptrN);
i = 100;
printf("ptrX -> %d\n", ptrX[100]);
```

Resolução:

Os erros e a sua possível correcção são os seguintes:

1. `ptrC = c;`
   Sendo `prtC` um apontador para uma zona capaz de conter um caracter, deveria ser inicializado a um endereço de um caracter, logo, uma possível correcção seria `ptrC = &c;`.
2. `ptrN = (int) malloc(10*sizeof(int));`
   Sendo `prtN` um apontador para uma zona capaz de conter um inteiro, dever-se-ia converter o endereço genérico devolvido pelo malloc para um endereço para inteiro pelo que, a instrução correcta seria `ptrN = (int*) malloc(10*sizeof(int));`
3. Quanto ao conjunto de instruções,

   ```
   i = 100;
   printf("ptrX -> %d\n", ptrX[100]);
   ```

   podemos supor que `ptrX` seria `ptrN` e que se pretendia aumentar o *array* de 10 para mais de 100 elementos(?). De acordo com este raciocínio, ter-se-ia:

   ```
   i = 101;
   ptrN = (int*) realloc(ptrN, i*sizeof(int));
   printf("ptrX -> %d\n", ptrX[100]);
   ```