Ficha de Trabalho N.º 4

Versão 2024/25

Objetivos: Utilização de Funções; Ponteiros, *Arrays* e *Strings*; Passagem de parâmetros por referência.

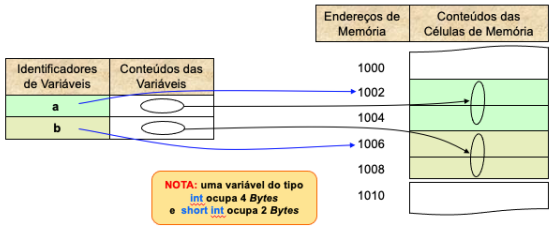
Conceitos necessários à resolução da ficha

Funções:

Os apresentados na ficha anterior, mais os seguintes:

Noção de variável, Endereço e Apontador

Como vimos na primeira aula de Algoritmos e Programação, quando declaramos uma variável estamos a alocar-lhe espaço na memória, em função do seu tipo.



Em C existe o operador & que permite saber qual o endereço de uma variável

O compilador associa a cada variável uma posição ÚNICA em memória, capaz de suportar os dados do tipo dessa variável. Sempre que num programa fazemos referência a uma variável, estamos na realidade a referir o conteúdo do endereço ou conjunto de endereços que essa variável ocupa.

Um apontador é uma variável que aponta sempre para outra variável, de determinado tipo.

Para indicar que uma variável é um apontador, coloca-se antes dela um asterisco (*).

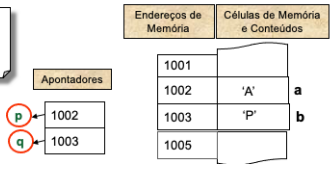
SINTAXE DE DECLARAÇÃO:

tipo *ptr

ptr - identificador da variável
tipo - tipo da variável para a qual apontará
* - indica que é uma variável do tipo apontador

Exemplo:

```
char a, b, *p = &a, *q = &b;
```



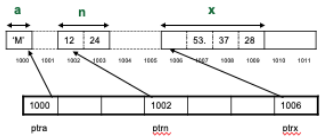
Apontadores e tipos de dados

Um apontador tem de possuir um determinado tipo: o tipo da variável para a qual aponta

O endereço de uma variável é sempre o menor dos endereços que ela ocupa em memória

Pelo facto de as variáveis ocuparem diferentes tamanhos em memória, os apontadores para essas variáveis necessitam de saber quantos Bytes de memória terão de considerar

```
char *ptra;
short int *ptrn;
float *ptrx;
```

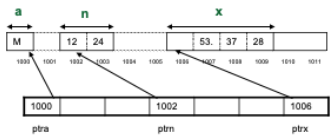


INICIALIZAÇÃO

A inicialização de um apontador faz-se, geralmente, através do operador & (Endereço de).

EXEMPLO

```
char *ptra = &a;
int *ptrn = &n;
float *ptrx = &x;
```



Um apontador para o tipo xyz endereça sempre o número de bytes que esse tipo ocupa em memória (portanto, endereça **sizeof**(xyz)).

NOTA: o operador sizeof devolve o número de Bytes ocupado por uma variável ou por um determinado tipo, numa dada arquitectura.

Sintaxe: sizeof <expressão> ou sizeof(<tipo>)

Aritmética de apontadores

Uma vez declarado um apontador, podem ser realizadas sobre ele praticamente todas as operações realizáveis sobre variáveis.

Exemplos:

```
int a = 8, b = 27;
int *ptr = NULL;// ptr não aponta para nada
ptr = &a; // ptr aponta para a
printf("Valores de a, b e *ptr: %d, %d, e %d", a,b,*ptr);// Valores de a, b e *ptr: 8, 27 e 8
*ptr = 50;//a variável cujo endereço está em ptr (a) recebe o valor 50
printf("a=%d, b=%d, *ptr= %d",a, b, *ptr); // é escrito a=50, b=27, *ptr=50
```

Operações com apontadores

Incremento

Um apontador para o tipo xyz avança sempre sizeof(xyz) Bytes por cada unidade de incremento.

Decremento

Um apontador para o tipo xyz recua sempre sizeof(xyz) Bytes por cada unidade de decremento.

Diferença

Quando realizável, permite saber quantos elementos existem entre um endereço e outro.

Comparação

É possível a comparação de dois apontadores para o mesmo tipo, usando os operadores relacionais

NOTA

As operações de diferença e comparação só podem ser realizadas entre apontadores do mesmo tipo

Apontadores e vectores

O nome de um vector corresponde sempre ao endereço do seu primeiro elemento:
se v for um vector, utilizar v é o mesmo que utilizar &v[0]

```
Exemplo
char s[]="Viva";
char *ptr = s;// ptr fica com o endereço de s[0] (&s[0])

Para acedermos ao carácter a presente na string, podemos ir por diversas vias
```

s[3]	Carácter existente na posição índice 3 do vector s
*(ptr+3)	Como ptr contém o endereço do primeiro carácter, prt+3 contém o endereço do 4º
*(s+3)	Notar que s==&s[0]
ptr[3]	O endereçamento através de parêntesis rectos pode usar-se também por apontadores

Passagem de vectores para funções

Se passarmos um vector v como argumento a uma função, o que ela recebe na realidade não é o vector, mas o endereço do seu primeiro elemento (v==&v[0]).

Para que uma função conheça o número de componentes dos vectores que lhe são passados, deve ser-lhe fornecido outro parâmetro indicador do número de elementos ou um terminador de cada vector.

OBSERVAÇÕES

Para evitar problemas de programação, é conveniente inicializar sempre os apontadores.

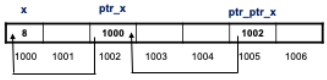
Quando se pretende declarar um apontador e não se quer que aponte para nenhuma variável, utiliza-se a variável simbólica NULL.

Numa declaração de um vector com inicialização automática (int *p = NULL) é o apontador p que é inicializado e não *p, embora a atribuição possa eventualmente sugerir o contrário.

Apontadores de apontadores (cont.)

Exemplo

Expressão	Tipo	Valor	Descrição
x	int	8	Valor da variável x
ptr_x	int *	1000	Endereço da variável x
*ptr_x	int	8	Valor apontado por ptr_x
ptr_ptr_x	int **	1002	Endereço do apontador ptr_ x
*ptr_ptr_x	int *	1000	Valor apontado por ptr_ptr_x
**ptr_ptr_x	int	8	Valor apontado pelo endereço apontado por ptr_ptr_x



Tipos de passagem de parâmetros

Passagem de parâmetros por valor: são enviadas para a função cópias dos valores de que esta necessita.

A passagem por valor permite, assim, enviar o valor de uma variável ou expressão.

Passagem de parâmetros por referência: o que é enviado para a função não são cópias dos valores, são as próprias variáveis.

Em C só existe passagem de parâmetros por valor: ao colocarmos uma variável como parâmetro, o que passa é o seu valor

Para conseguirmos alterar o valor de uma variável dentro de uma função, passamos o endereço da variável e efectuamos as alterações na variável original: USAMOS APONTADORES.

Passagem de vectores para funções

O nome de um vector é, em si mesmo, um endereço. Por isso, os vectores são sempre passados às funções sem o &.

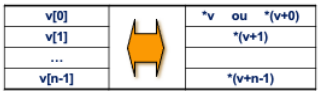
```
void leVector_va(float *u, int num)
// Funçao para leitura de um vector com num componentes reais
{  int i;
      for (i = 0; i < num; i++)
      {
            printf("Qual o %do elemento? ",i+1);
            scanf("%f", &u[i]);
      }
}
```

EQUIVALENTES

```
void leVector_vb(float *u, int num)
// Funçao para leitura de um vector com num componentes reais
{
      int i;
      for (i = 0; i < num; i++)
      {
            printf("Qual o %do elemento? ",i+1);
            scanf("%f", u+i);
      }
}
```

Apontadores e vectores

Se v tiver sido declarado como um vector com n componentes, v é um apontador para o primeiro elemento:



Para obtermos o elemento de índice i do vector v, pode utilizar-se:
v[i] ou *(v+i)

Apontadores de apontadores

Uma vez que os apontadores ocupam espaço em memória, é possível obter a sua posição através do operador &.

EXEMPLO

```
Consideremos uma variável x do tipo int:
      int x;
Se pretendermos armazenar o seu endereço, declaramos um apontador do tipo
da variável:
      int *prt_x;
Se pretendermos armazenar o endereço deste apontador, declaramos uma
variável do tipo do apontador  para o tipo de variável:
      int **prt_ptr_x;
```

Esse processo de endereçamento podia prosseguir consecutivamente.

Envio do endereço de uma variável para uma função

Questão

Quando é que se torna necessário passar endereços de variáveis para uma função?

Se a variável for um *array*, a resposta é NUNCA, já que o nome do vector corresponde ao endereço do seu primeiro elemento.

Se a variável não for um *array*, tem de ser precedida de & sempre que se pretender que a própria variável seja alterada dentro da função ou procedimento a que se destina.

Se passarmos o endereço de um *array*, a variável que o recebe tem de ser um apontador para o tipo dos elementos do *array*.

Passagem de vectores para funções

```
void escreveVector_va(float *u, int num)
// Funçao para escrita de um vector com num componentes reais
{
      int i;
      for (i = 0; i < num; i++)
            printf("\t [%d]=%.1f   ", i+1, u[i]);
}
```

EQUIVALENTES

```
void escreveVector_vb(float *u, int num)
// Funçao para escrita de um vector com num reais
{
      int i;
      for (i = 0; i < num; i++)
            printf("\t [%d]=%.1f   ", i+1, *(u+i));
      printf("\n");
}
```

Problemas Propostos

1 - Implemente uma função que permita trocar o valor de duas variáveis. Teste a função num pequeno programa para o efeito.

2 - Elabore um programa que leia uma string e escreva os dois primeiros caracteres no monitor.

3 - Escreva uma função, utilizando ponteiros, que, dadas duas strings (cadeias de caracteres) *str1* e *str2*, como parâmetros de entrada, faça a sua concatenação, devolvendo a string resultante em *str1*. Por exemplo, se *str1* for "Aula" e *str2* for "Pratica" a função deverá devolver "AulaPratica" em *str1*.

4 - Elabore funções, utilizando ponteiros, que:

a) Leia um vetor de n elementos inteiros (n é especificado pelo utilizador);
b) Apresente no monitor os elementos do vetor;
c) Apresente no monitor os elementos do vector pela ordem inversa;
d) Determine o máximo do vetor;
e) Determine o mínimo do vetor;
f) Determine a soma das componentes;
g) Determine a média dos elementos do vetor;
h) Determine quantos elementos do vetor são superiores à média e quantos são inferiores;
i) Duplique para outro vetor os valores lidos;
j) Intercale os dois vetores (o 2.º por ordem inversa) e disponibilize um terceiro vetor resultante. Ex.:

vetor *x* = [1 3 5 7]
=> vetor final = [1 8 3 6 5 4 7 2]
vetor *y* = [2 4 6 8]