UNIVERSIDADE da MADEIRA

CTeSP CURSOS TÉCNICOS SUPERIORES PROFISSIONAIS

TECNOLOGIAS DE PROGRAMAÇÃO DE SISTEMAS DE INFORMAÇÃO

Ponteiros ou Apontadores- Precauções

PROGRAMAÇÃO ORIENTADA A OBJECTOS| Prof. Doutora Frederica Gonçalves

Cofinanciado por:

MADEIRA 14-20

REGIÃO AUTÓNOMA DA MADEIRA

PORTUGAL 2020

UNIÃO EUROPEIA
Fundo Social Europeu

- **Ponteiros e *strings***
  - Verificou-se anteriormente que quando utilizamos *strings* como ***arrays* de char** podemos definir uma *string* **t** da seguinte forma:

```
char t [ ] = " Aula";
```

  - Sendo definida na memória como:

t [0]	t [1]	t [2]	t [3]	t [4]
'A'	'u'	'l'	'a'	'\0'

UNIDADE CURRICULAR : PROGRAMAÇÃO ORIENTADA A OBJECTOS

- **Ponteiros e *strings***
  - De igual forma, podemos definir a variável **t** como se segue:

```
char * t = " Aula";
```

  - Nesta declaração a variável **t**, é declarada como um **ponteiro para char.**

*t	*(t+1)	*(t+2)	*(t+3)	*(t+4)
'A'	'u'	'l'	'a'	'\0'

UNIDADE CURRICULAR : PROGRAMAÇÃO ORIENTADA A OBJECTOS

Consideremos agora dois programas que funcionam da mesma maneira, escrevendo a *string* **t (“Aula”)** no ecrã.

Declaração como *array*

```
#include <iostream>
using namespace std;

main () {
char t[ ] = “Aula";
cout << t << ‘\n’;
}
```

Declaração como *ponteiro*

```
#include <iostream>
using namespace std;

main () {
char *t = " Aula ";
cout << t << ‘\n’;
}
```

UNIDADE CURRICULAR : PROGRAMAÇÃO ORIENTADA A OBJECTOS

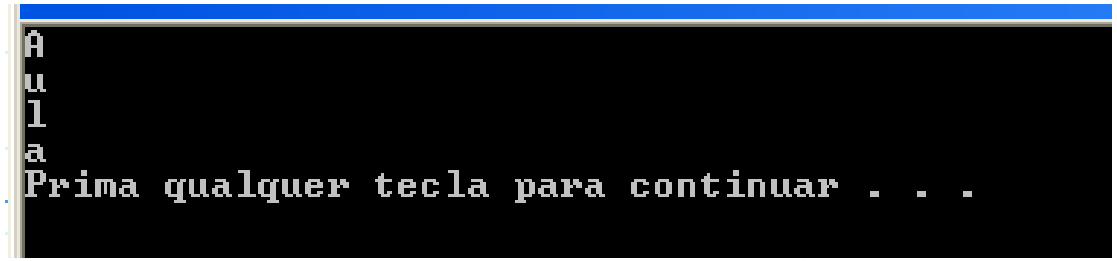
Queremos agora escrever, carácter a carácter a *string* **t** como um ponteiro, utilizando o ciclo **for.**

```
#include <iostream>
using namespace std;

main ( )
{
   char *t = “Aula";
   for (int i=0; *(t+i) != '\0'; i++)
   cout <<*(t+i)<<'\n';
   system (“Pause”);
}
```

UNIDADE CURRICULAR : PROGRAMAÇÃO ORIENTADA A OBJECTOS

Resultado no ecrã



- Usou-se a variável **i** como contador do ciclo.
- A expressão ***(t+i),** permite a incrementação de ***i*** dentro do ciclo, percorrendo os vários valores da **string t.**
- A condição de controlo do ciclo é: ***(t+i) != '\0'**.

UNIDADE CURRICULAR : PROGRAMAÇÃO ORIENTADA A OBJECTOS

Vejamos agora a *string* **t** escrita como um ponteiro, carácter a carácter, utilizando o ciclo **while.**

```
#include <iostream>
using namespace std;

main ( )
{
   char *t = “Aula";
   while (*t != '\0')
   cout <<*t++<<'\n';
}
```

- Usou-se o ponteiro **t** com o operador de incremento ***t++** para escrever os caracteres da *string*, não necessitando da variável **i** como contador.
- A condição de controlo do ciclo é: ***(t != ‘\0’).**

UNIDADE CURRICULAR : PROGRAMAÇÃO ORIENTADA A OBJECTOS

Vejamos outra forma da *string* **t** ser escrita como um ponteiro, carácter a carácter, utilizando o ciclo **while.**

```
#include <iostream>
using namespace std;

main ( )
{
char *t = “Aula de BP";
while (*t)
cout <<*t++<<'\n';
}
```

- A variável **t** é declarada como um ponteiro para **char**, podendo receber assim diferentes números de caracteres.
- A instrução utiliza o ciclo **while** e a expressão ***t++** para percorrer os caracteres da *string* até ao fim.

- **Ponteiros e *strings***
  - Quando uma ***string* t** é declarada como ponteiro e utilizando um ciclo **while** reduz-se o código e tornamos o programa mais eficiente.

  - Uma outra vantagem da utilização de ponteiros na declaração de *strings*, prende-se ao facto de estas poderem ser declaradas sem inicialização e receberem *strings* com diferentes tamanhos.

- **Algumas precauções a ter em conta na utilização de ponteiros**

  - A utilização de **ponteiros** (ou apontadores), confere uma maior flexibilidade e eficiência de escrita de programas, mas no entanto, representam um maior risco em termos de ocorrência de erros, e uma maior dificuldade na sua detecção.
  - Isto porque, o compilador não detecta nem denuncia grande parte dos erros.

UNIDADE CURRICULAR : PROGRAMAÇÃO ORIENTADA A OBJECTOS

Utilizar um ponteiro com **variáveis de um tipo** de dados que **não correspondem** ao tipo de dados com que esse ponteiro foi declarado.

```
#include <iostream>
using namespace std;
main ()
{
float  x = 1.25;
int *p; p = &x;
cout << *p << "\n";
}
```

Qual é o erro?

- ***p** declarado para inteiro (**int** *p)
- É-lhe atribuído o endereço **x**
- O endereço **x** é do tipo **float**.

UNIDADE CURRICULAR : PROGRAMAÇÃO ORIENTADA A OBJECTOS

Utilizar um **ponteiro antes de o ter inicializado** com endereço de alguma variável.

```
#include <iostream>
using namespace std;
main ()
{
float  x = 10, *p;
*p = x;
cout << *p << "\n";
}
```

Para onde aponta p ?

- **x** declarado para inteiro
- ***p** é declarado para inteiro
- Porém a instrução **(*p=x)** não faz sentido.
- Se **(*p=x)** fosse substituída por **p=&x**, funcionaria.

UNIDADE CURRICULAR : PROGRAMAÇÃO ORIENTADA A OBJECTOS

Escrever **instruções com erros** de lógica **nas operações** de aritmética com ponteiros.

```
#include <iostream>
using namespace std;
main ()
{
int  x [3] = {10, 20, 30};
cout << *x<< "\n";
cout << *x++ << "\n";
}
```

O que está mal ?

- **cout << *x** é um ponteiro constante
- **cout << *x++** não funciona pois pretende-se incrementar x como ponteiro variáve

UNIDADE CURRICULAR : PROGRAMAÇÃO ORIENTADA A OBJECTOS

Escrever **instruções com erros** de lógica **na precedência** dos operadores com ponteiros.

Quando utiliza-se, numa mesma expressão os operadores:

- ✓de inderecção **(*)**
- ✓de incremento **(++)** ou de decremento **(--)**

Deve ter em conta:

Que eles têm a mesma prioridade ou precedência, mas a sua associatividade faz-se da direita para a esquerda.

UNIVERSIDADE da MADEIRA

CTeSP CURSOS TÉCNICOS SUPERIORES PROFISSIONAIS