Ponteiros

Strings Constantes – Modificador de Acesso *const*

UNIVASF

Ponteiros

- Inicializando Ponteiros

Podemos inicializar, ponteiros de um jeito, no mínimo interessante.

Para isto, precisamos entender como a linguagem de programação C trata as strings constantes.

Toda string constante que o programador insere no programa é colocada num banco de strings que o compilador cria. No local onde está uma string constante no programa, o compilador coloca o endereço do início desta string no banco de strings constantes.

UNIVASF

String Constante 1

String Constante 2

Banco de strings contantes

strcmp (char *, char *)

strcmp (“EU”, “VAMOS”)

Ponteiros

- Inicializando Ponteiros

O que isto tem a ver com a inicialização de ponteiros?

É que, para uma string constante que vamos usar várias vezes, podemos fazer:

```
char *str1="String constante.";
```

Aí poderíamos, em todo lugar que precisarmos da string, usar a variável **str1**. Devemos apenas tomar cuidado ao usar este ponteiro. Pois, se o alterarmos vamos perder a referência para a string e se o usarmos para alterar a string podemos facilmente corromper o banco de strings constantes que o compilador criou.

UNIVASF

Ponteiros

- Inicializando Ponteiros

OBS.: Em C existem modificadores de acesso, um exemplo é o modificador *const* que permite criar uma constante.

Exemplo:

```
const int numero = 32;
```

Logo, podemos fazer:

```
const char *str1="String constante.";
```

Desta forma, o endereço apontado pelo ponteiro não pode ser alterado. Mas, ainda podemos corromper o banco de strings constantes.

UNIVASF

Ponteiros

Funções strchr() e strstr() - string.h

UNIVASF

Ponteiros

Agora podemos concluir nosso estudo das Funções Básicas para manipulação de Strings.

- strchr

Sua forma geral é:

*char *strchr (const char *str, int ch);*

A função *strchr()* devolve um ponteiro para a primeira ocorrência do byte menos significativo de *ch* na string apontada por *str*. Se não for encontrada nenhuma coincidência, será devolvido um ponteiro nulo. (string.h)

UNIVASF

Ponteiros

/* Exemplo strchr */

```c
#include <string.h>
int main()
{
    char *p;
    p = strchr("Isto eh um teste.",' ');
    puts(p);
}
```

Ponteiros

- strstr

Sua forma geral é:

*char *strstr (const char *str1, const char *str2);*

A função *strstr*() devolve um ponteiro para a primeira ocorrência da string apontada por *str2* na string apontada por *str1*. Ela devolve um ponteiro nulo se não for encontrada nenhuma coincidência. Obs.: Função presente em string.h.

UNIVASF

Ponteiros

/* Exemplo strstr */

```c
#include <string.h>
int main()
{
    char *p;
    p = strstr("Isto é um teste", "te");
    puts(p);
}
```

Ponteiros

Exercício:

Construa um programa que receba duas strings fornecidas pelo usuário, verifique se a segunda string está presente na primeira e, caso esteja retorne a posição do caractere na primeira string onde a primeira ocorrência da segunda string inicia, caso contrário retorne zero.

UNIVASF

```c
#include <stdio.h>
#include <string.h>
#include <stdlib.h>
int main() {
    char string1[100], string2[100], *p;
    puts("Forneca a primeira string: ");
    scanf("%99[^\n]", string1);
    puts("Forneca a segunda string: ");
    setbuf (stdin, NULL);
    scanf("%99[^\n]", string2);
    p = strstr(string1, string2);
    if (p)
        printf("%d\n", (int)(p-string1)+1);
    else
        puts ("0");
}
```

Vetores de Ponteiros e Ponteiros para Ponteiros

UNIVASF

Ponteiros

- Vetores de ponteiros

Podemos construir vetores de ponteiros como declaramos vetores de qualquer outro tipo.

Um exemplo de declaração de um vetor de ponteiros inteiros é:

```
int *pmat [10];
```

No caso acima, **pmat** é um vetor que armazena 10 ponteiros para inteiros.

UNIVASF

Ponteiros

- Ponteiros para Ponteiros

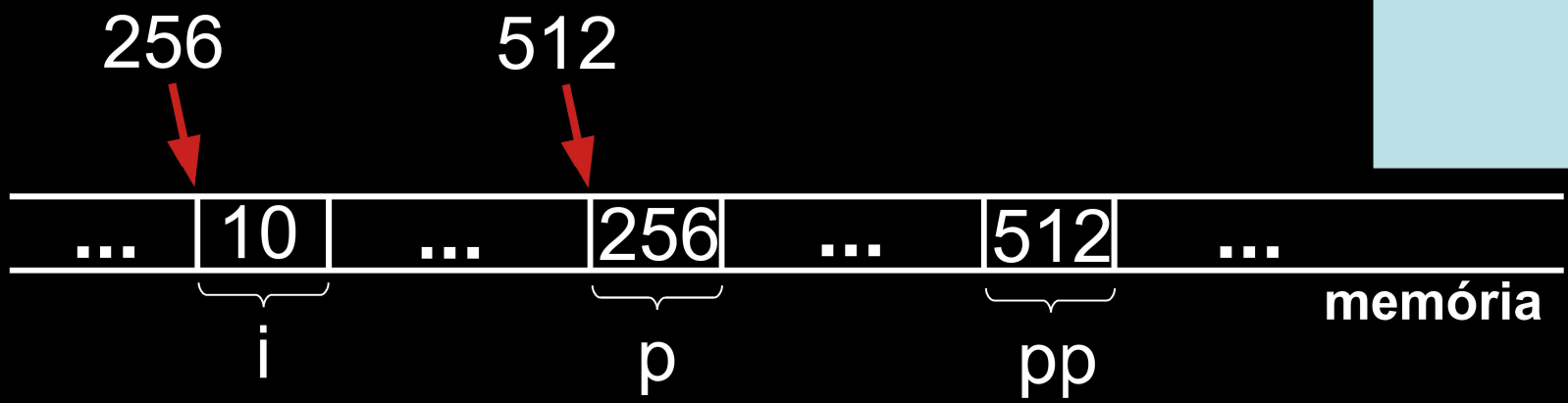
Podemos declarar um ponteiro para um ponteiro com a seguinte notação:

*tipo_da_variável **nome_da_variável;*

Algumas considerações:

****nome_da_variável** é o conteúdo final da variável apontada;

***nome_da_variável** é o conteúdo do ponteiro intermediário.




```
int i=10;
int *p;
int **pp;
p=&i;
pp=&p;
```

```
printf ("%d", **pp);
```

Ponteiros

- Ponteiros para Ponteiros (continuação)

Na linguagem C podemos declarar ponteiros para ponteiros para ponteiros, ou então, ponteiros para ponteiros para ponteiros para ponteiros e assim por diante.

Para fazer isto basta aumentar o número de asteriscos na declaração.

Para acessar o valor desejado apontado por um ponteiro para ponteiro, o operador asterisco deve ser aplicado duas vezes, como mostrado anteriormente e no exemplo a seguir:

UNIVASF

Ponteiros

- Ponteiros para Ponteiros (continuação)

```c
#include <stdio.h>
int main()
{
    float pi = 3.1415, *pf, **ppf;
    pf = &pi;
    ppf = &pf;
    printf("\n%.4f", **ppf);
    printf("\n%.4f", *pf);
}
```

Ponteiros

Exercício:

Verifique o programa abaixo. Encontre o(s) seu(s) erro(s) e corrija-o(s) para que o mesmo escreva o número 10 na tela.

```
#include <stdio.h>
int main()
{
      int x, *p, **q;
      p = &x;
      q = &p;
      x = 10;
      printf("\n%d\n",**q);
}
```

Funções

Funções

Funções são as estruturas que permitem ao usuário separar seus programas em blocos (subprogramas). Para fazermos programas grandes e complexos devemos construí-los bloco a bloco.

Uma função na linguagem C tem a seguinte forma geral:

```
tipo_de_retorno nome_da_função (declaração_de_parâmetros)
{
        corpo_da_função
}
```

Funções

O tipo-de-retorno é o tipo do valor que a função vai retornar. O default é o tipo **int**, ou seja, o tipo-de-retorno assumido por omissão. A declaração de parâmetros é uma lista com a seguinte forma geral:

tipo nome1, tipo nome2, ... , tipo nomeN

Observe que o tipo deve ser especificado para cada uma das N variáveis de entrada. É na declaração de parâmetros que informamos ao compilador quais serão as entradas da função (assim como informamos a saída no tipo-de-retorno).

É no corpo da função que as entradas são processadas, a saída é gerada ou outras operações são executadas.

UNIVASF

Funções

- Comando return

Forma geral:

return valor_de_retorno; ou *return;*

Quando se executa uma declaração **return** a função é encerrada imediatamente e, se o valor de retorno é informado, a função retorna este valor. É importante lembrar que o valor de retorno fornecido tem que ser compatível com o tipo de retorno declarado para a função.

UNIVASF

```c
#include <stdio.h>
int Square (int a)
{
    return (a*a);
}
int main ()
{
    int num;
    printf ("\nEntre com um numero: ");
    scanf ("%d",&num);
    num=Square(num);
    printf ("\n\nO seu quadrado vale: %d\n",num);
}
```

```c
#include <stdio.h>
int Square (int a)
{
    return (a*a);
}
int main ()
{
    int num;
    printf ("\nEntre com um numero: ");
    scanf ("%d",&num);
    printf ("\n\nO seu quadrado vale: %d\n",
    Square(num));
}
```

Funções

Observação:

Devemos nos lembrar que a função **main()** é uma função e como tal devemos tratá-la. A função **main()** retorna um inteiro. Isto pode ser interessante se quisermos que o sistema operacional receba o valor de retorno da função **main()**. Se assim o quisermos, devemos nos lembrar da seguinte convenção: se o programa retornar zero, significa que ele terminou normalmente, e, se o programa retornar um valor diferente de zero, significa que o programa teve um término anormal.

UNIVASF

```c
#include <stdio.h>
int EPar (int a)
{
     if (a%2)
          return 0;
     else
          return 1;
}
int main ()
{
     int num;
     printf ("Entre com numero: ");
     scanf ("%d",&num);
     if (EPar(num))
          printf ("\n\nO numero e par.\n");
     else
          printf ("\n\nO numero e impar.\n");
     return 0;
}
```

```c
return (!(a%2));
```

Funções

Exercício

Construa um programa que possua a função "EDivisivel(int **a**, int **b**)", escrita por você. A função deverá retornar 1 se **a** for divisível por **b**. Caso contrário, a função deverá retornar zero. O programa deve ler dois números fornecidos pelo usuário (**a** e **b**, respectivamente), e utilizar a função EDivisivel para retornar uma mensagem dizendo se **a** é ou não divisível por **b**.

UNIVASF