UniEVANGÉLICA
UNIVERSIDADE EVANGÉLICA DE GOIÁS

Aula 02

Estruturas Estáticas x Estruturas Dinâmicas

Variáveis Dinâmicas

Ponteiros

Alocação Dinâmica de Memória

Prof.(a): William P. Santos Júnior

Estruturas Estáticas:

- Até agora estudamos a estrutura de dados que tem um tamanho pré-definido. Estruturas como Arrays (Vetores e Matrizes) e Structs. Isso ocorre pois o compilador aloca – designa – de forma automática o espaço necessário de memória para cada uma dessas estruturas. Podemos dizer com isso que alocação de memória estática ocorre no momento da compilação;

```c
#include <stdio.h>

typedef struct {
    char *nome;
    int idade;
    float nota;
} Pessoa;

void imprimirDadosAluno(Pessoa discente);
void definirDadosAluno(Pessoa *discente, char nome[], int idade, float nota );

int main() {

    Pessoa discente1, discente2;


    definirDadosAluno(&discente1, "William", 42, 8.5);
    imprimirDadosAluno(discente1);

    definirDadosAluno(&discente2, "Zeze", 32, 9.1);
    imprimirDadosAluno(discente2);

    return 0;
}

void imprimirDadosAluno(Pessoa discente) {

    printf("\nDados do aluno\n");

    printf("Nome: %s\n", discente.nome);
    printf("Idade: %d\n", discente.idade);
    printf("Nota: %.1f\n", discente.nota);
}

void definirDadosAluno(Pessoa *discente, char nome[], int idade, float nota) {
    discente->nome = nome;
    discente->idade = idade;
    discente->nota = nota;
}
```

Estruturas Dinâmicas:

- Nesse tipo de alocação de espaços de memória, podemos definir o espaço necessário durante a execução de um programa, ou seja, a alocação é feita em tempo de execução. Isso é muito interessante pois, permite que o espaço em memória seja alocado somente quando necessário e ainda permite o aumento e a diminuição da quantidades de memória alocada no programa.

```c
#include<stdio.h>
#include<stdlib.h>
#include<locale.h>

main()
{
  setlocale(LC_ALL,"Portuguese");

  float *v;
  int i, TamVetor;

  v = (float * ) malloc(TamVetor * sizeof(float));

  printf("Informe o número de componentes do vetor\n");
  scanf("%d", &TamVetor);

  for (i = 0; i < TamVetor; i++)
  {
   printf("\nDigite um valor para a %dª Posição do vetor: ", i+1);
   scanf("%f",&v[i]);
  }
 // ------ Percorrendo o vetor e imprimindo os valores ----------
 printf("\n********** Valores do vetor dinamico **********\n\n");

 for (i = 0;i < TamVetor; i++)
 {
   printf("\t\t\t%.2f\n",v[i]);
 }
 free(v);
}
```

Alocação Dinâmica de Memória:

- Alocação dinâmica ocorre em tempo de execução:
- Utilizada quando não sabemos o quanto de espaço será necessário para o armazenamento dos dados que estamos trabalhando:
- Dessa forma, o espaço de memória utilizado será somente o necessário e evita desperdício de memória do computador.

As funções **malloc** e **free,** e o operador ***sizeof,*** são essenciais para a alocação de memória.

Alocação Dinâmica de Memória:

- ***malloc***, usa o número de bytes a serem alocados consecutivamente como argumento/parâmetro, e retorna um ponteiro do tipo ***void*** para a memória alocada. Um ponteiro ***void**** pode ser atribuído a uma variável de qualquer tipo de ponteiro. A função ***malloc*** geralmente é usada como o operador ***sizeof.***

onde:

m: memory
alloc: alocation

```
v = (float * ) malloc(TamVetor * sizeof(float));
```

- O operador ***sizeof*** retorna a quantidade de bytes de um determinado tipo de dados, ex: ***x = sizeor(int)***, neste caso a variável ***x*** receberá o valor 4, que é correspondente ao número de bytes de um dado do tipo ***int***.

- A função ***free***, libera o espaço de memória alocado.

Alocação Dinâmica de Memória:

- Na linguagem C, a alocação dinâmica de memória pode ser realizada com apenas quatro chamadas a funções:

  - void * malloc(int qty_bytes_alloc);
  - void * calloc(int qty, int size);
  - void * realloc(void * pointer, int new_size);
  - free(void * pointer);

- A função **malloc** permite que seja feita a alocação de uma nova área de memória para uma estrutura.
- A função **calloc** tem a mesma funcionalidade de malloc, exceto que devem ser fornecidos o tamanho da área e a quantidade de elementos.
- A função **realloc** permite que uma área previamente alocada seja aumentada ou diminuída e a função free libera uma área alocada previamente com a função malloc, calloc ou realloc.

Alocação Dinâmica de Memória:

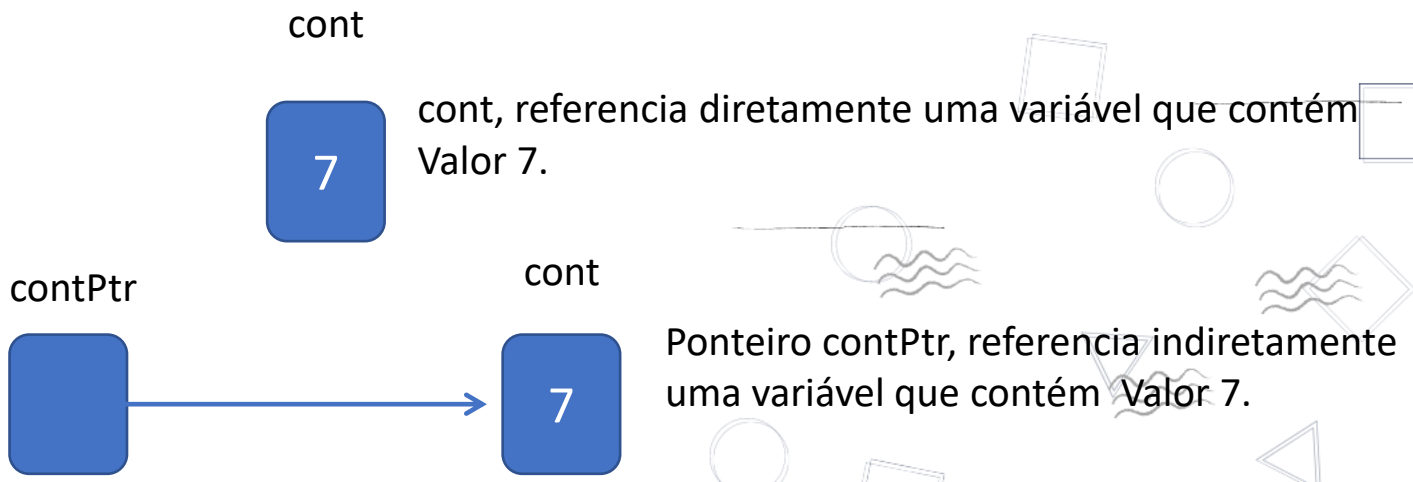
Mas antes de continuarmos com alocação de memória, relembraremos o que são ponteiros.

Ponteiros:

O que são ponteiros?
Um dos recursos mais poderosos da Linguagem C e também uma das capacidades mais difíceis de se dominar em C, entretanto eles permitem que os programas simulem uma chamada por referencia, criem e manipulem estruturas dinâmicas de dados.

Declaração: ***int*** **contPtr cont;*

*Onde: *contPtr, é uma variável do tipo ponteiro para um tipo int, ou seja, essa variável aponta para um objeto do tipo int. O que define, então, uma variável ponteiro é o sinal inserido antes do nome da variável, (*).*



Ponteiros:

Operadores de Ponteiros

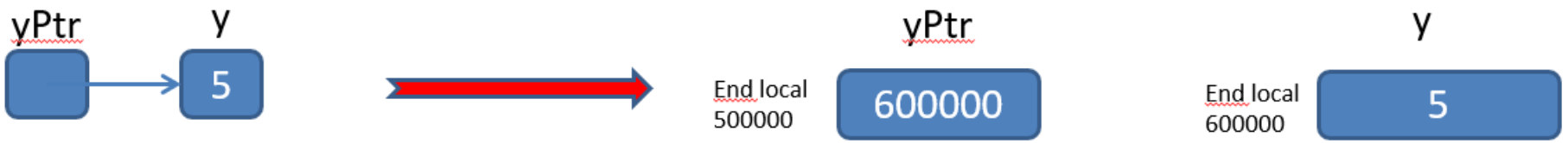
&: operador de endereço, é um operador unário(pesquisar), que retorna o endereço de seu operando. Por exemplo, considerando as seguintes definições:

```
int y = 5;
int *yPtr;
```

a instrução:

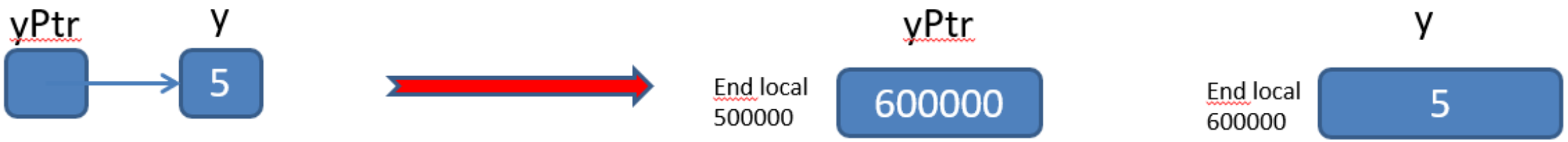
```
yPtr = &y;
```

atribui o endereço de ***y*** na variável ***yPtr,*** então, podemos dizer que a variável ***yPtr*** aponta para a variável ***y*.**



Ponteiros:

Operadores de Ponteiros:



Essa figura representa o ponteiro na memória, supondo que a variável indireta *y* esteja armazenada no endereço de memória 600000, e a variável de ponteiro esteja no endereço 500000. O operador unário *, retorna o valor do objeto apontado por seu operando, ou seja, o ponteiro.

Imprimindo o conteúdo da variável y: `printf("%i", *yPtr);`

Ponteiros:

Operadores de Ponteiros:

O código ao lado demonstra os operadores de ponteiro & e *. O especificador de tipo %p de printf (), mostra o local de memória como um inteiro hexadecimal(pesquisar) na maioria das plataformas.

Pesquisar:

Como passar argumentos/parâmetros para funções por referencia?

```c
#include<stdio.h>
#include<stdlib.h>
#include<locale.h>
main()
{
  setlocale(LC_ALL,"Portuguese");

  int a; // a é uma variável inteiro
  int *aPtr; // aPtr é um ponteiro para um inteiro

  a    = 7;
  aPtr = &a; //aPtr definido para o endereço de a;

  printf("\n O endereço de a é %p \
          \n O valor de aPtr é %p",&a,aPtr);

  printf("\n O Valor de a é %i \
          \n O valor de aPtr é %i",a,*aPtr);

  printf("\n\n Mostrando que * e & são complemento um \
          \n do outro \n\n   &*aPtr = %p \
          \n   *&aPtr = %p.",&*aPtr,*&aPtr);

}
```

Atividades:

1. Escreva um programa em linguagem C que solicita ao usuário a quantidade de alunos de uma turma e aloca um vetor de notas (números reais). Depois de ler as notas, imprime a média aritmética. Obs: não deve ocorrer desperdício de memória; e após ser utilizada a memória deve ser devolvida.
2. Faça um programa que leia do usuário o tamanho de um vetor (inteiros) a ser lido e faça a sua alocação dinâmica de memória. Depois, leia do usuário seus valores e imprima o vetor lido e mostre quantos dos números são pares e quantos são ımpares.
3. Faça um programa que receba do usuário o tamanho de uma string e chame uma função para alocar dinamicamente essa string. Em seguida, o usuário deverá informar o conteúdo dessa string. O programa imprime a string sem suas vogais.
4. Crie um programa que declare uma estrutura (registro) para o cadastro de alunos. a) Deverão ser armazenados, para cada aluno: matrıcula, nome (apenas um) e ano de nascimento. b) Ao início do programa, o usuário deverá informar o número de alunos que serão armazenados c) O programa devera alocar dinamicamente a quantidade necessária de memória para armazenar os registros dos alunos. d) O programa deverá pedir ao usuário que entre com as informações dos alunos. e) Ao final, mostrar os dados armazenados e liberar a memória alocada.

UniEVANGÉLICA

UNIVERSIDADE EVANGÉLICA DE GOIÁS