ESTRUTURA DE DADOS

AULA 07 – ALOCAÇÃO DINÂMICA

ALOCAÇÃO DE MEMÓRIA

- Alocar memória significa reservar espaço em memória;
- Um conjunto de bytes é reservado para um programa de modo que este possa utilizar para armazenar os dados que serão necessários;
- Esta alocação é solicitada ao SO pelo programa que entrará em execução. Caso não haja atendimento desta solicitação o programa não "entrará" na memória e não será executado.
- Existem duas formas de alocação de memória:
  - Alocação estática;
  - Alocação dinâmica.

ALOCAÇÃO DE MEMÓRIA

- Existem “três” modos de reservar espaço de memória:

1ª - Estática: uso de variáveis globais

- O espaço reservado na memória para uma variável global existe enquanto o programa estiver sendo executado e somente é liberado quando o programa sai da execução.

2ª - Estática: uso de variáveis locais

- O espaço reservado na memória existe apenas enquanto o módulo (função ou procedimento) que declarou a variável está sendo executada, sendo liberado para outros usos quando a execução da função termina.

Em ambos os casos, as variáveis foram alocadas estaticamente, pois em suas declaração não foram utilizadas funções de alocação dinâmica.

ALOCAÇÃO DE MEMÓRIA

- **Até agora, os programas utilizavam a memória do computador estaticamente: todas as posições de memória eram reservadas para as variáveis no início da execução do programa ou da função e, mesmo que não estivessem sendo mais utilizadas, continuavam reservadas para as mesmas variáveis até a conclusão da execução do programa ou da função.**
- **Exemplo:** ***Um vetor global do tipo float com mil componentes, por exemplo, "ocupará" quatro mil bytes de memória durante toda a execução do programa (considerando-se que cada float ocupa 4 bytes). Naturalmente, isto pode, em grandes programas, sobrecarregar ou, até mesmo, esgotar a memória disponível. No primeiro caso, há uma degradação na eficiência do programa; no segundo caso a execução do programa pode ser inviabilizada.***

ALOCAÇÃO DINÂMICA: POR QUE???

- **Imagine a seguinte situação:**

Situação: Precisamos construir um programa que processe os valores dos salários dos funcionários de uma pequena empresa.

Solução: Declarar um array do tipo float bem grande com, por exemplo, umas 1.000 posições como float salarios[1000];

Dificuldades:

- Se a empresa tiver menos de 1.000 funcionários esse array será um exemplo de desperdício de memória, pois nem todas as posições serão utilizadas;

- Se a empresa tiver mais de 1.000 funcionários esse array será insuficiente para lidar com os dados de todos os funcionários. Logo, o programa não atende as necessidades da empresa.

- Em ambos os casos, deve-se alterar o código do programa, mais isto não significa que futuras mudanças quanto ao tamanho do vetor não seja novamente necessárias.

ALOCAÇÃO DINÂMICA

3ª - Dinâmica: uso de variáveis globais ou locais

- Requisita-se, em tempo de execução, espaço de um determinado tamanho (alocado dinamicamente) que permanece reservado até que explicitamente seja liberado pelo programa;
- A solicitação do espaço desejado deve ser explicitada por alguma função de alocação durante a codificação do programa;
- A solicitação de espaço pode ser realizada em instantes de tempo diferentes e a quantidade de bytes alocada está diretamente relacionada ao tipo de dado;
- Ao liberar espaço, este será disponibilizado para outros usos e não pode mais ser acessado;
- Se o não for explicitada a liberação de espaço, este será automaticamente liberado quando a execução do programa terminar.

ALOCAÇÃO DINÂMICA

- As posições de memória são reservadas para variáveis no instante em que são necessárias e essas posições são liberadas para o sistema, nos instantes em que não estejam sendo utilizadas, por meio de funções específicas para isso;
- Permite reservar espaço de memória de tamanho arbitrário (qualquer) e acessá-los através de apontadores.
- Permite escrever programas mais flexíveis, pois nem todos os espaços alocados devem ser definidos/fixados ao escrever no código do programa.
- O C permite alocar dinamicamente (em tempo de execução), blocos de memória usando ponteiros. Ou seja, o programador cria, em tempo de execução, a reserva de novos espaços em memória, evitando-se desperdício de memória.

ALOCAÇÃO DINÂMICA

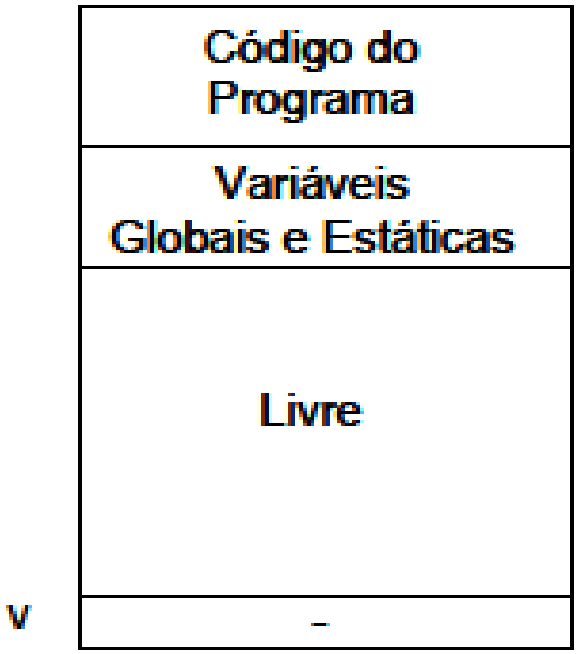
- A alocação dinâmica consiste em **requisitar** um **espaço** de **memória** ao computador em **tempo** de **execução**, o qual **devolve** para o programa o **endereço** do **início desse espaço alocado** usando um **ponteiro**.

ALOCAÇÃO DINÂMICA

- **O que ocorre na memória durante a alocação dinâmica, está representado abaixo.**

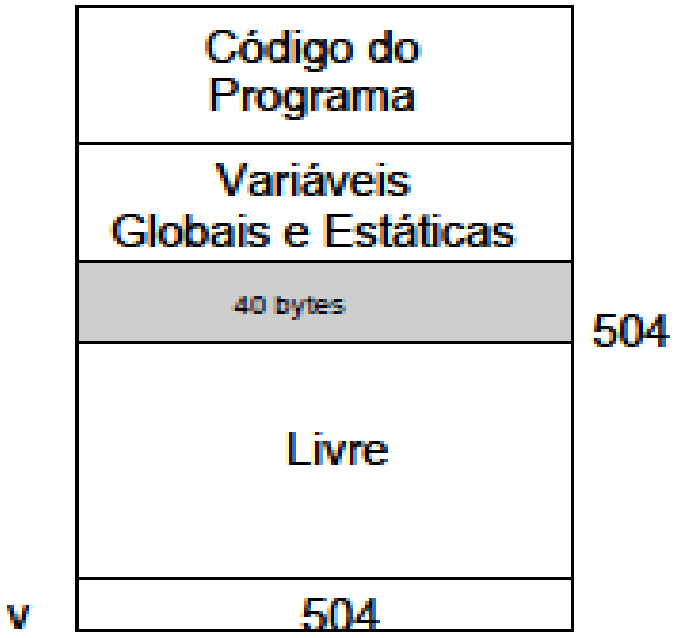
1 - Declaração: int *v

Abre-se espaço na pilha para o ponteiro (variável local)



2 - Comando: v = (int *) malloc (10*sizeof(int))

Reserva espaço de memória da área livre e atribui endereço à variável



ALOCAÇÃO DINÂMICA: FUNÇÕES

- **A alocação e liberação desses espaços de memória é feito por algumas funções padrão da biblioteca *<stdlib.h>*.**
- ***malloc()* – aloca um determinado espaço de memória.**
- ***calloc()* - aloca um determinado espaço de memória.**
- ***realloc()* - para expandir a memória necessária, após o uso da função *malloc()*.**
- ***free()* – libera um determinado espaço de memória.**

FUNÇÃO *MALLOC()*

- Abreviação de *memory allocation*.
- Aloca um bloco de bytes consecutivos na memória e devolve para um ponteiro o endereço do primeiro bloco de memória reservado/alocado.
- O bloco alocado é maior que o solicitado. Os bytes adicionais são usados para guardar informações administrativas sobre esse bloco e permitem que ele seja corretamente desalocado, mais tarde.
- Retorna um ponteiro *void** (genérico), para qualquer tipo desejado.
- Deve-se utilizar um *type cast* (modelador) para transformar o ponteiro devolvido para um ponteiro do tipo de dado desejado.

FUNÇÃO *MALLOC()*: SINTAXE

- Quando a função *malloc()* é executada é feita uma solicitação de memória ao SO e caso obtenha sucesso o espaço de memória é reservado. Ou seja, um endereço de bloco reservado é retornado pela função e será armazenado na variável ponteiro.
- Como a memória não é infinita, caso não haja memória suficiente disponível no momento da solicitação, a função retorna como resultado *NULL*.
- No momento da alocação da memória, deve-se levar em conta o tamanho do dado alocado, pois cada tipo de dado ocupa uma quantidade de bytes diferente.

FUNÇÃO *MALLOC()*: SINTAXE

- **Sintaxe geral:**

```
<ponteiro>=(<tipo_dado>*) malloc(sizeof(<tipo_dado>));
```

- **Permite aumentar a eficiência de utilização de memória pelo fato de que a quantidade de memória alocada será aquela que é suficiente para armazenar o dado, sem que haja desperdício, podendo "crescer" à medida da necessidade (sob demanda).**
- **A função *malloc* retorna um ponteiro para *void* (1 byte), por isso, geralmente, é acompanhada do *type cast* de acordo com o tipo de dado que desejado.**

FUNÇÃO *SIZEOF()*: SINTAXE

- **A função sizeof() é usada para saber o tamanho em bytes de variáveis ou de tipos.**
- **Sintaxes:**

sizeof nome_da_variável

sizeof (nome_do_tipo)

```
1 #include <stdio.h>
2 #include <stdlib.h>
3 struct ponto{
4     int x,y;
5 };
6 int main(){
7   printf(''Tamanho char: %d\n'',sizeof(char));
8   printf(''Tamanho int: %d\n'',sizeof(int));
9   printf(''Tamanho float: %d\n'',sizeof(float));
10  printf(''Tamanho double: %d\n'',sizeof(double));
11  printf(''Tamanho struct ponto: %d\n'',sizeof(struct
        ponto));
12  int x;
13  double y;
14  printf(''Tamanho da variavel x: %d\n'',sizeof x);
15  printf(''Tamanho da variavel y: %d\n'',sizeof y);
16  system(''pause'');
17  return 0;
18 }
```

Variações da sintaxe de função *malloc()*: Sintaxe1

```
pont = (tipo *)malloc(tam);

pont = (tipo*)malloc(num*sizeof(tipo));
```

cast

onde:

pont é o nome do ponteiro que recebe o endereço do espaço de memória alocado.

tipo é o tipo do endereço apontado (tipo do ponteiro).

tam é o tamanho do espaço alocado: numero de *bytes*.

num é o numero de elementos que queremos poder armazenar no espaço alocado.

sizeof() retorna o número de bytes de um inteiro.

```
<ponteiro>=(<tipo_dado>*) malloc(sizeof(<tipo_dado>));
```

Função *malloc()*: exemplos

```
int *p;
p = (int*) malloc(n * sizeof(int));
```

```
ptr = (int*) malloc (1000*sizeof(int));
if (ptr == NULL)
{
   printf ("Sem memoria\n");
   return 1;
}
```

```
typedef struct {
   int dia, mes, ano;
} data;

data *d;

d = malloc( sizeof (data));

d->dia = 31; d->mes = 12; d->ano = 2008;
```

- **Função malloc(): exemplos. Alocar 20 caracteres para conter uma *string*.**

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

int main(void)
{
    char *nome;
    nome = (char *) malloc(21);

    printf("Digite seu nome: ");
    gets(nome);

    printf("%sn", nome);

    return 0;
}
```

Podemos declarar e inicializar o ponteiro na mesma linha.

```c
char *nome = (char *) malloc(21);
```

```c
char *nome = (char *) malloc(21*sizeof(char));
```

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

int main(){
   int *p;
   p = (int *) malloc(5*sizeof(int));
   int i;
   for (i=0; i<5; i++){
      printf(''Digite o valor da posicao %d: '',i);
      scanf(''%d'',&p[i]);
   }
   system(''pause'');
   return 0;
}
```

- ***Aloca** um **array** contendo 5 posições de inteiros.*
- *sizeof(int) retorna 4 (número de bytes do tipo int na memória), perfazendo um total de 20 bytes alocados.*
- *malloc() retorna um ponteiro genérico, que é convertido para o tipo do ponteiro via type cast: (int*).*
- *o ponteiro p passa a ser tratado como um **array**: p[i].*

```c
#include <stdio.h>
#include <stdlib.h>
int main(){
   int *p;
   p = (int *) malloc(5*sizeof(int));
   if(p == NULL){
      printf(''Erro: Memoria Insuficiente!\n'');
      exit(1);
   }
   int i;
   for (i=0; i<5; i++){
      printf(''Digite o valor da posicao %d: '',i)
          ;
      scanf(''%d'',&p[i]);
   }
   system(''pause'');
   return 0;
}
```

Melhorando o código do programa:

1. ***Deve-se sempre testar se foi possível fazer a alocação de memória.***
2. ***Quando malloc() retorna um ponteiro NULL isto indica que não há memória disponível no computador, ou que algum outro erro ocorreu que impediu a memória de ser alocada.***

FUNÇÃO *MALLOC()*

- **Alocar dinamicamente um vetor de inteiro com 10 elementos. Ex.:**

```
int *v;

v = (int *) malloc(10*sizeof(int));
```

- ***Precisa-se armazenar valores inteiros na área alocada dinamicamente por malloc, foi declarado um ponteiro de inteiro para receber o endereço inicial do espaço alocado. Depois, podemos tratar v como tratamos um vetor declarado estaticamente, pois, se v aponta para o inicio da área alocada, podemos dizer que v[0] acessa o espaço para o primeiro elemento que armazenaremos, v[1] acessa o segundo, e assim por diante, até v[9].***

PRIMEIROS PASSOS:

Implemente dinamicamente o vetor para as situações, abaixo:

1º :

Criar um vetor cujo tamanho será informado pelo usuário em tempo de execução;

2º :

Inicializar o vetor com valores fornecidos pelo usuário;

3º :

Exibir todos valores da última até a 1ª posição;

4º :

Calcular e exibir a quantidade de números pares;

5º :

Calcular e exibir a média de todos os valores.