PROGRAMAÇÃO DE COMPUTADORES V - TCC- 00.323

Modulo 12: alocação dinâmica de memória

Aura - Erick
aconci@ic.uff.br, erickr@id.uff.br

Roteiro

- porque e como utilizar a alocação dinâmica
- funções:
  - **malloc ( ) ,**
  - **calloc ( ) ,**
  - **realloc ( )** e
  - **free ( ) .**
- Trabalho 8

A alocação é ***estática***

As declarações abaixo alocam espaço de memória 3 variáveis.

A **alocação é *estática*** (nada a ver com a palavra-chave **static**), ou seja, acontece **antes que o programa comece a ser executado**:

```
char c;
int i;
int v[10];
```

funções **malloc e free de stdlib**.

Às vezes, a quantidade de memória a alocar só se torna conhecida **durante a execução do programa.**

Para lidar com essa situação é preciso recorrer à **alocação *dinâmica* de memória**.

A alocação dinâmica é gerenciada pelas funções **malloc e free**, que estão na biblioteca **stdlib**.

Para usar esta biblioteca, você deve incluir a correspondente interface no seu programa por meio de

#include <stdlib.h>

Alocação dinâmica

é o processo que obter memória em tempo de execução (quando o programa está sendo usado).

Ela é utilizada quando **não se sabe ao certo quanto de memória será necessário para o armazenamento das informações**, podendo ser determinadas em tempo de execução conforme a necessidade do programa.

Dessa forma evita-se o desperdício de memória.

No C padrão

existem 4 funções para alocações dinâmica pertencentes a biblioteca **stdlib.h** .

São elas :

malloc ( ) ,

calloc ( ) ,

realloc ( ) e

free ( ) .

Sendo mais utilizadas são **malloc ( )** e **free ( ) .**

As funções **malloc()** e **calloc()** são responsáveis por alocar memória, a **realloc()** por realocar a memória e por último a **free()** fica responsável por liberar a memória alocada.

A alocação dinâmica é muito utilizada em casos que usam muito volume de dados.

A sintaxe da função **malloc()** é :

void * malloc (size_t size)

esta função **recebe** como parâmetro "**size**" que é o **número de bytes de memória** que se deseja alocar.

O tipo **size_t** é definido em **stdlib.h** como sendo um inteiro sem sinal.

Esta função retorna **um ponteiro** do **tipo void** podendo assim ser atribuído a qualquer tipo de ponteiro.

Como se faz?

Suponha que seja necessário no meio do código alocar uma memória com 1500 bytes, para isto seria necessário apenas digitar as seguintes linhas de código:

```
cha * str ;
str = malloc ( 1500 ) ;
```

(o nome é uma abreviatura de *memory allocation*)

A função malloc

aloca um bloco de bytes consecutivos na memória do computador e devolve o endereço desse bloco.

O número de bytes é especificado no argumento que voce vai passar para a função.

O endereço devolvido por malloc é do tipo genérico:

void *.

O programador , você, armazena esse endereço num ponteiro de tipo apropriado, para seu programa e caso!

Alocação dinâmica é muito utilizada em casos que **não se sabe antes o volume** de dados.

```c
#include <stdio.h>
#include<stdlib.h> /* para utilizar o system("Pause") */

void main(void)
{
  int QuantElem, i;
  int vet[99];

  printf("Digite a quantidade de valores que deseja inserir no vetor:");
  scanf("%d",&QuantElem);

  /* Colhendo os dados a serem colocados no vetor:*/
  printf("\nDigite os valores a serem inseridos no vetor vet:\n");
  for (i=0;i<QuantElem;i++)
  {
    printf("Digite o valor de vet[%d]:",i);
    scanf("%d", &vet[i]);
  }

  /* Imprimindo os dados do vetor na tela:*/
  printf("\nImprimindo os valores do vetor vet:\n");
  for (i=0;i<QuantElem;i++)
 printf("vet[%d]:%d\n",i,vet[i]);

  system("Pause");
}
```

Quando não se sabe antes a quantidade de elementos a serem utilizados deve-se utilizar a alocação dinâmica.

O exemplo anterior é muito simples, mas mostra em que situação se deve utilizar alocação dinâmica.

Vemos que **não se sabe a quantidade de valores** que vai ser inserido no vetor, por esse motivo declaramos um vetor de 99 posições.

Mas e se a pessoa, por exemplo, deseja colocar apenas 3 elementos no vetor. Como o vetor foi declarado com o tamanho igual a 99, o que acontece é o desperdício de memória.

Mas, e caso passe de 100 haverá um estouro.

Portanto a melhor solução é utilizar alocação dinâmica.

O mesmo exemplo mostrado anteriormente com o uso de alocação dinâmica (usando a função malloc()) :

```c
#include <stdio.h>
#include<stdlib.h>
#include <alloc.h> /* Para o uso da alocação dinâmica (malloc() e free()) */
#include<conio.h> /* Para o uso do getch() */

void main(void)
{
   int QuantElem, i;
   int *vet; /* uso de ponteiro */

   printf("Digite a quantidade de valores que deseja inserir no vetor:");
   scanf("%d",&QuantElem);

  /*Alocando memoria para o vetor vet do tamanho determinado pelo usuário,
  desta forma não terá diperdicio de memoria".*/
  if ((vet = (int *) malloc(QuantElem)) == NULL)
   {
     printf("Memoria insuficiente");
     exit(1);
   }

   /* Colhendo os dados a serem colocados no vetor:*/
   printf("\nDigite os valores a serem inseridos no vetor vet:\n");
   for (i=0;i<QuantElem;i++)
   {
     printf("Digite o valor de vet[%d]:",i);
     scanf("%d", &vet[i]);
   }

   /* Imprimindo os dados do vetor na tela:*/
   printf("\nImprimindo os valores do vetor vet:\n");
   for (i=0;i<QuantElem;i++)
      printf("vet[%d]:%d\n",i,vet[i]);

   free(vet);

   getch();
}
```

Heap é a região de memória livre do seu computador.

Agora o usuário pode digitar o tamanho do vetor que quiser que **não haverá desperdício de memória** e o vetor a ser alocado será do tamanho digitado pelo usuário.

A função **malloc()** utilizada no programa , devolve um ponteiro do tipo **void**, desta forma pode-se ser atribuído **a qualquer tipo de ponteiro** (no exemplo é atribuído a um ponteiro do tipo inteiro).

A memória alocada pela função **malloc(),** como também por outras funções com o mesmo fim que vamos ver no decorrer desta aula, deve ser obtida do **heap**.

Há espaço ?

É necessário verificar se a memória livre (heap) é suficiente para o armazenamento e isto pode ser feito como mostrado na segunda seta no exemplo anterior :

```
if ((vet = (int *) malloc(QuantElem)) == NULL)
 {
   printf("Memoria insuficiente");
   exit(1);
 }
```

Caso a função **malloc()** devolver o **valor nulo** significa que não há memória suficiente e nesse caso o programa apresenta na tela a mensagem : "**Memória insuficiente**" e imediatamente termina o programa através da função **exit().**

Perigo!!!

- As variáveis alocadas estaticamente dentro de uma função desaparecem quando a execução da função termina.
- Já as variáveis alocadas dinamicamente **continuam a existir mesmo depois que a execução da função termina**.
- Se for necessário liberar a memória ocupada por essas variáveis, é preciso recorrer à função free.

free ( ) ;

A função **free ( )** usada neste exemplo , na terceira seta, e que você deve usar com as demais funções de alocação dinâmica serve para liberar a memória alocada.

E sua sintaxe é a seguinte:

void free (void *p) ;

O **p** é um ponteiro que aponta para a memória alocada anteriormente, no programa anterior, o ponteiro **vet**.

dangling pointers ??

- A função free libera a porção de memória alocada. A instrução free (ptr) avisa ao sistema que o bloco de bytes apontado por ptr está livre. A próxima chamada de alocação poderá tomar posse desses bytes.
- A função free não deve ser aplicada a uma *parte* de um bloco de bytes alocado: aplique free apenas ao bloco todo.

- Convém não deixar ponteiros soltos (= *dangling pointers*) no seu programa, pois isso pode ser explorado por hackers para atacar o seu computador.
- Portanto, depois de cada **free (ptr),** atribua NULL a ptr:
  - free (ptr); ptr = NULL;

OBS.

- Cada invocação de **malloc** aloca um bloco de bytes consecutivos maior que o solicitado: os bytes adicionais são usados para guardar informações administrativas sobre o bloco de bytes (essas informações permitem que o bloco seja corretamente “desalocado”, mais tarde, pela **free** .
- O número de bytes adicionais pode ser grande, e maior que o número de bytes solicitado no argumento de **malloc**.
- Não é ineficiente, portanto, alocar pequenos blocos de bytes; é preferível alocar um grande bloco por vez.

calloc ( )

Esta função também tem como objetivo alocar memória e possui a seguinte sintaxe:

void *calloc (unsigned int num , unsigned int size);

num representa a quantidade de memória a ser alocada e **size** é o seu tamanho.

Esta função inicia o espaço alocado com **0**.

Podemos usar, no exemplo anterior, **calloc ( )** só alterando na segunda seta:

O mesmo exemplo:

```
#include <alloc.h>
#include<conio.h>

void main(void)
{
   .
   .
   .

  /*Alocando memoria para o vetor vet do tamanho determinado pelo usuário,
  desta forma não terá diperdicio de memoria".*/
  if ((vet = (int *) calloc(QuantElem, sizeof(int))) == NULL)
  {
     printf("Memoria insuficiente");
     exit(1);
  }

   .
   .
   .

   free(vet);

   getch();
}
```

A função **sizeof( )** retorna o tamanho em bytes do tipo de dado, que no nosso exemplo é **inteiro**, int, isto é importante para que o **programa seja portável**.

realloc ( )

Esta função é responsável por **realocar** a memória cuja a sintaxe é:

void * realloc (void * **p**, **size_t** **size**)

Desta forma pode-se alterar o tamanho da memória anteriormente alocado por ***p** para o tamanho especificado por um novo valor (**size**).

exemplo de realloc ( )

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

void main(void)
{
   char *string;

   /* alocando memoria com malloc() */
   string = (char *) malloc(6);

   /* copiando "Julio" para string: */
   strcpy(string, "Julio");
   /* Imprimindo a string: */
   printf("string:%s\n",string);

   /* realocando string, agora com o tamanho 20:*/
   string = (char *) realloc(string, 20);

  /* copiando "Julio Battisti" para string:*/
   strcpy(string, "Julio Batistti\n");

   /* Imprimindo o novo valor da string: */
   printf("Nova String:%s\n",string);

   /* liberando memória: */
   free(string);

   system("Pause");
}
```

Obs.

Neste último programa usamos a função **strcpy()** que copia a string entre aspas para a string destino, que no nosso caso é uma variável com o nome de string.

Para utilizar esta função foi necessário inserir no cabeçalho **“#include <string.h>”.**

Nos exemplos mostrados **system(“Pause”)** tem o mesmo papel que o **getch()** e serve para que o programa dê uma pausa.

Para se utilizar o **system(“Pause”)** é necessário inserir no cabeçalho **“#include<stdlib.h>”.**

Referencias

http://www.ime.usp.br/~pf/algoritmos/

http://juliobattisti.com.br/tutoriais/katiaduarte/cbasico009.asp

http://www.ime.usp.br/~pf/algoritmos/aulas/aloca.html

http://www.inf.puc-rio.br/~inf1007/material/slides/alocacaodinamica.pdf

http://www.di.ufpb.br/liliane/aulas/alocacao.html

http://www.inf.ufpr.br/cursos/ci067/Docs/NotasAula/notas-32_Aloca_c_cao_dinamica_mem.html

http://linguagemc.xpg.uol.com.br/aloc_dinamica.html

https://www.youtube.com/watch?v=FD0bi4H08IU

https://pt.wikipedia.org/wiki/Gerenciamento_de_mem%C3%B3ria

8º Trabalho - Entrega: 11 / 03 /2016

Escreva duas novas versões do **7º** Trabalho onde seja usada :

malloc ( ) e **free ( ) ,**

calloc ( ) e **free ( )** ,

para um vetor que tenha em cada elemento a estrutura "**produtos_supermercado**"

e cujo **número de elementos** seja definido pelo usuário apenas em **tempo de execução.**