Alocação de Memória Estática & Dinâmica

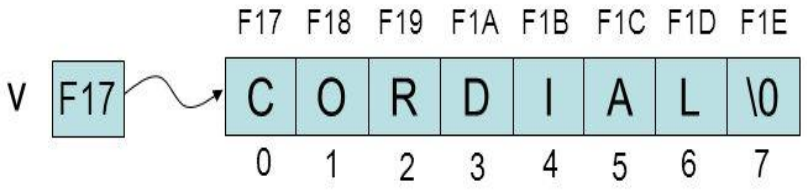
ALOCANDO MEMÓRIA

Tudo o que o computador faz exige armazenamento em memória, ou seja, não processa sem que os dados sejam previamente armazenados na memória. Na memória são armazenados, programas, funções, variáveis, constantes, ponteiros, objetos, estruturas, pilhas, listas, filas, árvores, grafos, arquivos, banco de dados, documentos, etc.

Por exemplo, um vetor de caracteres com oito posições seria armazenado na memória de acordo com a figura abaixo:

"Entenda, o nome de um vetor nada mais é do que um ponteiro apontando para sua primeira posição"

```
char v[8];
```



- Portanto, uma outra forma de declarar o vetor é:

```
char *v;
```

ALOCAÇÃO ESTÁTICA

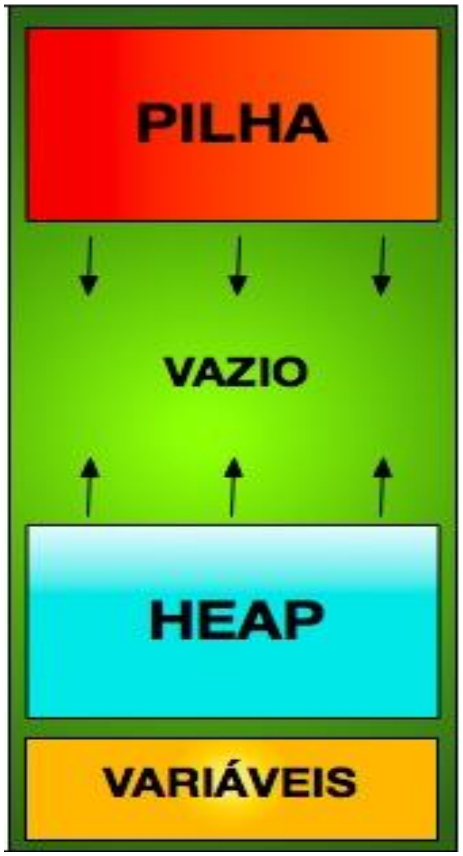
Neste tipo de alocação, os tipos de dados tem tamanho predefinido, onde o compilador irá alocar de forma automática o espaço de memória necessário. Sendo assim, dizemos que a alocação estática é feita em tempo de compilação. Este tipo de alocação tende a desperdiçar recursos, já que nem sempre é possível determinar previamente qual é o espaço necessário para armazenar as informações. Quando não se conhece o espaço total necessário, a tendência é o programador exagerar pois é melhor superdimensionar do que faltar espaço.

EXEMPLO: double lista [100];

Ao se declarar um vetor com 100 posições, não é garantido que no programa as 100 posições serão preenchidas, então as posições não preenchidas ficarão ocupando espaço na memória, por isso, dizemos que na alocação estática é comum ocorrer desperdício de memória.

ALOCAÇÃO DINÂMICA

É o processo de solicitar e utilizar memória durante a execução de um programa. Ela é utilizada para que um programa em C utilize apenas a memória necessária para sua execução, sem desperdícios de memória. Isto é bem interessante, pois permite que o espaço em memória seja alocado apenas quando necessário. Além disso, a alocação dinâmica permite aumentar ou até diminuir a quantidade de memória alocada.



A pilha (**Stack**) armazena endereços de retornos de funções, parâmetros, variáveis globais, constantes, etc.

O pilha **Heap**, é a região de memória livre onde podemos alocar dinamicamente memória para objetos, ponteiros, ou seja, estruturas encadeadas, árvores, grafos, etc.

Funções Sizeof(), malloc() e free( )

A função **sizeof** determina o número de bytes para um determinado tipo de dados. É interessante notar que o número de bytes reservados pode variar de acordo com o compilador utilizado. Basicamente a função receberá o tipo de dado como argumento e retorna o tamanho em bytes de memória que irá ocupar este tipo.

EXEMPLO1 - usando o comando sizeof :

```
#include "iostream"
#include "cstdlib"
using namespace std;
int main () {
cout << "Um tipo int     tem " <<sizeof(int) << " bytes " << endl;
cout << "Um tipo float   tem " <<sizeof(float) << " bytes " << endl;
cout << "Um tipo double  tem " <<sizeof(double) << " bytes " << endl;
cout << "Um tipo char    tem " <<sizeof(char) << " bytes " << endl;
cout << "Um tipo bool    tem " <<sizeof(bool) << " bytes " << endl;
system("pause"); return 0; }
```

Funções malloc() e free( )

A função **malloc** é utilizada para se alocar memória para o armazenamento de um dado qualquer, de um determinado tipo. Por isso deve-se utilizar ponteiros que devem apontar para o endereço de memória reservado ou alocado. No caso do C é necessário um comando para desalocar a memória, para isso serve o comando **free**, liberar o espaço de memória alocado.

EXEMPLO2 - Alocando memória para um inteiro

```
#include "iostream"
#include "cstdlib"
using namespace std;
int main () {  setlocale (LC_ALL, "Portuguese");
int * ptx = (int *)  malloc( sizeof ( int ) ); // aloca 4 bytes para um inteiro

if (ptx == NULL) cout << "\nNão foi possível a alocação de memória!";
else  { *ptx = 10;
 cout << "O valor " << *ptx << " será armazenado na memória " << ptx << endl;
}

system("pause");
free (ptx);  // liberar a memória
return 0; }
```

Alocando memória para um STRUCT

Um struct após ser criado se torna um novo tipo de dado no código, portanto, você poderá utilizar o comando sizeof() e o comando malloc para forçar a alocação dinâmica para para criar novos structs como se fossem objetos na memória. No caso do C, é necessário um comando para desalocar a memória, para isso serve o comando **free**, liberar o espaço de memória alocado.

EXEMPLO3 - Alocando memória para um STRUCT

```
#include "iostream"
#include "cstdlib"
using namespace std;
typedef struct me x;
struct me { int a; float b; char g;  };
int main () {  setlocale (LC_ALL, "Portuguese");
x * ptx =  (x *)  malloc( sizeof ( x ) ); // aloca 4 bytes para um inteiro

if (ptx == NULL) cout << "\nNão há mais memória!";
else   {
ptx->a = 10;    ptx->b=10.4;     ptx->g='0';
cout << "O struct ptx será foi armazenado na memória " << ptx << endl;
cout << "Valores:"<< ptx->a << "   " << ptx->b << "    "<< ptx->g << endl; }
system("pause");   free (ptx);  // liberar a memória
return 0; }
```

Encadeamento de Structs (Nó)

Utilizando um ponteiro e alocação de memória é possível criarmos várias cópias de um struct e ligar-mos umas às outras através dos ponteiros, isso se chama encadeamento de estruturas ou nós. Cada nó abaixo é um struct, que contém ponteiros (cad * **ligação** ou **liga**) que servem para armazenar o endereço do próximo nó da lista de nós. O último ponteiro liga ou ligação apontará para NULL neste exemplo.

Encadeamento de Structs (Nó)

Neste exemplo os nomes dos atributos **POSIÇÃO** e **LIGAÇÃO** foram trocados para **pos** e **liga**, o struct foi apelidado por **cad** para ficar igual ao desenho anterior.

EXEMPLO4 - Amarrando os Nós

```
#include "iostream"
#include "cstdlib"
using namespace std;
typedef struct me cad;
int cont=0;
struct me { int pos;  cad * liga;  };
cad * ultimoNo;  cad * primeiroNO;

void alocar () { cont ++;
 cad * newcad = (cad*) malloc ( sizeof ( cad ) );
if ( cont == 1 )  {  newcad->liga = NULL;
  newcad->pos=cont;
  primeiroNO=newcad;
  ultimoNo = newcad;}
else
{ ultimoNo->liga = newcad;
 newcad->liga=NULL;
 newcad->pos=cont; }
 ultimoNo=newcad; }
```

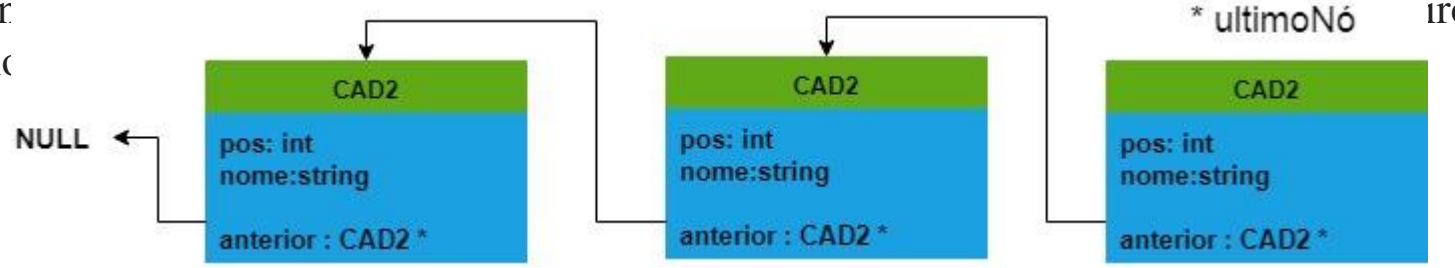
```
void exibir() {   cad * temp;   temp =
primeiroNO;
while ( temp != NULL) { cout <<
temp->pos << endl;
 temp = temp->liga; }   }

 void destruir() {  cad * temp;  temp =
primeiroNO;
while ( temp->liga != NULL)
{ free(temp); temp = temp->liga; }

free (primeiroNO); free (ultimoNo);}

 int main() { for (int i=0;i<=4;i++ )
alocar(); exibir(); destruir (); }
```

ADO Avaliação Contínua

I - Faça um menu para testar o programa 4 e faça as alterações a seguir:

A. Altere o código de C++ para C.
B. Insira mais um atributo no struct para armazenar um número aleatório criado por uma função randomica
C. Crie uma função que busque e exiba uma posição qualquer um dos nódulos da estrutura.
D. Crie função que informe se a estrutura está vazia.

II - Projete uma estrutura chamada **cad2** encadeada que contenha nódulos com os seguintes atributos: int pos; string nome; *cad2 anterior; A estrutura deverá ser encadeada da seguinte forma: Deverá ter um ponteiro chamado UltimoNo que para manter sempre o endereço de memória do ultimoNo que será adicionado na estrutura. Cada nódulo deverá ter um ponteiro interr iro nódulo adici



NOTA: *Trata-se de um desafio, claro que dúvidas surgirão, mas o importante é o aluno fazer todas as sub rotinas possíveis, caso não funcione corretamente o professor irá ajudá-lo durante o processo e também na próxima aula onde o assunto será retomado.*