Programação 2

Alocação Dinâmica

Rivera

Alocação Dinâmica

- Uso da memória
  - Uso de variáveis globais (e estáticas)
    - Espaço reservado enquanto o programa estiver ativo
  - Uso de variáveis locais
    - Espaço reservado apenas enquanto a função proprietária estiver ativa
  - Variáveis globais ou locais
    - Simples ou vetores
    - Vetor precisa número máximo de elemento
      - Compilador precisa calcular o espaço para reservar

A alocação dinâmica otimiza o uso de memória pelo programa, fazendo com que cada variável tenha seu espaço reservado durante apenas o tempo necessário

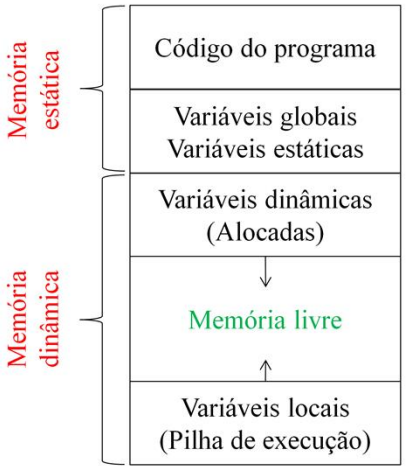
Alocação Dinâmica

- Uso da memória
  - Alocação dinâmica
    - Espaço de memória requisitado em tempo de execução
    - Espaço permanece reservado até que seja explicitamente liberado
      - Espaço disponível depois de liberado
      - Espaço alocado e não liberado explicitamente
        - » Automaticamente liberado ao final da execução

Ao fazer uso da alocação dinâmica, o espaço de memória é reservado até que seja explicitamente limpo (por meio da função free), ou até o final da execução do programa

Alocação Dinâmica

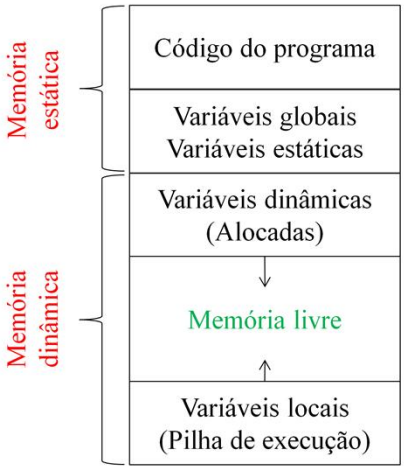
- Uso da memória
  - Memória estática
    - Código do programa
    - Variáveis globais
    - Variáveis estáticas
  - Alocação dinâmica
    - Variáveis dinâmicas (alocadas)
    - Memória livre
    - Variáveis locais



Variáveis estáticas e globais são armazenadas dentro da memória estática, enquanto variáveis locais e dinâmicas são alocadas dentro da memória livre, que por sua vez, fica dentro da memória dinâmica

Alocação Dinâmica

- Uso da memória
  - Alocação dinâmica de memória
    - Usa a memória livre
    - Se espaço de memória livre for menor que o requisitado
      - Não aloca. Tratar erro.
  - Pilha de execução
    - Alocação de memória quando ocorre chamada de função
      - Sistema reserva o espaço para variáveis locais da função
      - Liberado o espaço quando termina a função
    - Espaço não disponível (pilha cresce)
      - Programa abortado com erro



A alocação dinâmica utiliza da memória livre, logo, caso o espaço livre de memória seja menor que o requisitado, há erro. Já a pilha de execução

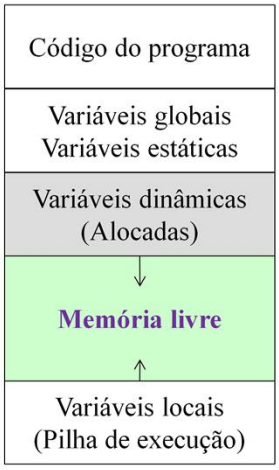
Alocação Dinâmica

- Funções da biblioteca padrão "stdlib.h"
  - Contém funções pré-determinadas
    - Funções para tratar alocação dinâmica de memória

      void * malloc (int num_bytes);

      void * calloc (int num, int num_bytes);

      void free (v);
    - Constantes pré-determinadas
    - ....



a biblioteca stdlib.h adiciona várias funções para alocação de memória

Alocação Dinâmica

- Função

  void * malloc (int num_bytes)

  - Retorna ponteiro genérico
    - Ponteiro convertido para o tipo apropriado
      - x = malloc (10 * sizeof (int)); // int * x;
    - Ponteiro convertido explicitamente
      - x = (int *) malloc (10 * sizeof(int)); // int * x;
    - Retorna endereço nulo (NULL)
      - Se não houver espaço
  - Operador "sizeof"
    - Retorna o número de bytes ocupado por um tipo
  - Função "free"
    - Parâmetro ponteiro de memória

A função malloc possibilita a alocação dinâmica de ponteiros, enquanto o operador sizeof retorna o número de bytes ocupado por um tipo, e a função free limpa a memória utilizada por uma variável

Alocação Dinâmica

- Exemplo:
  - Alocação dinâmica de um vetor de 10 elementos
    - Malloc retorna o endereço da área alocada
    - Ponteiro recebe endereço inicial do espaço alocado

```
int *v;
v = (int *) malloc ( 10 * sizeof (int))
If ( v == NULL)
{
     prontf (“ erro na alocação de 10 * int);
     exit();
}
```

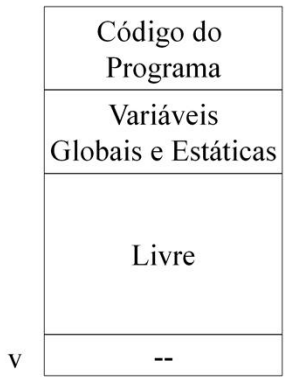
a função malloc é utilizada, acima, para a alocação dinâmica de memória de um vetor de 10 elementos, que após isso é associado a um ponteiro V, que recebe o endereço inicial do espaço alocado

Alocação Dinâmica

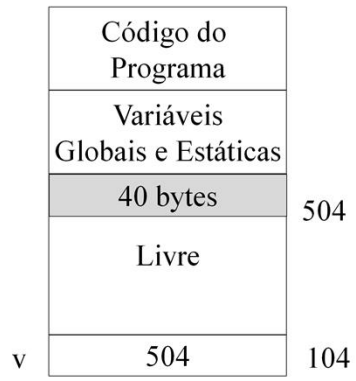
- Exemplo (cont.):

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

1: Declaração **int *v**
Abre-se espaço na pilha
para o ponteiro (var local)



2: Alocação:
v = (int *) malloc (10*sizeof(int))
Reserva espaço de memória da área
livre e atribui endereço à variável



Após a declaração de um ponteiro, abre-se um espaço de memória para ele, que depois de alocado, passa a armazenar um endereço, correspondente ao tamanho do vetor, que é alocado na memória livre

```c
#include <stdlib.h>
int main ( void )
{
   float *v;
   float med, var;
   int i,n;
   printf("Entre n e depois os valores\n");
   scanf("%d",&n);
   v = (float *) malloc(n*sizeof(float));
   if (v==NULL) { printf(“Falta memoria\n”); exit(1); }

   for ( i = 0; i < n; i++ )
       scanf("%f", &v[i]);

   med = media(n,v);
   var = variancia(n,v,med);

   printf ( "Media = %f Variancia = %f \n", med, var);
   free(v);
   return 0;
}
```

primeiramente, é incluída a stdlib.h, após isso, o ponteiro v é declarado, e depois recebe o endereço alocado dinâmicamente de um vetor do tipo float, de tamanho n decidido pelo usuário. No final da execução, o espaço de memória ocupado por v é limpo, utilizando a função free

Exercícios:

1. Dado um conjunto de arquivos de texto. Estabelecer uma ordem de acesso aos arquivos dependendo do número de caracteres, exceto mais de dois espaços, que eles contêm.
2. Ler de um arquivo N nomes de pessoas e suas respectivas idades. Agrupar as pessoas por idade e mostrar nomes por cada idade.
3. Existem um conjunto de N pontos no espaço 3D que representam vértices de faces poligonais (4 lados) de um objeto. Deseja-se pintar as faces do objeto com uma cor (R,G,B) proporcional ao ângulo definido entre ao vetor normal de cada face e o vetor de raio de luz que está na origem do sistema cartesiano. O vetor normal de uma face é computado como produto vetorial de dois vetores arestas que tem um pronto (vértice) de origem comum.