Alocação Dinâmica

Introdução à Computação

Alocação de memória

- **Uso da memória:**
- uso de variáveis globais (e estáticas):
  - O espaço reservado para uma variável global existe enquanto o programa estiver sendo executado.

- uso de variáveis locais:
  - Neste caso, o espaço existe apenas enquanto a função que declarou a variável está sendo executada, sendo liberado para outros usos quando a execução da função termina. Assim, a função que chama não pode fazer referência ao espaço local da função chamada.

Alocação dinâmica de memória

- A linguagem C oferece meios de requisitarmos espaços de memória em tempo de execução.

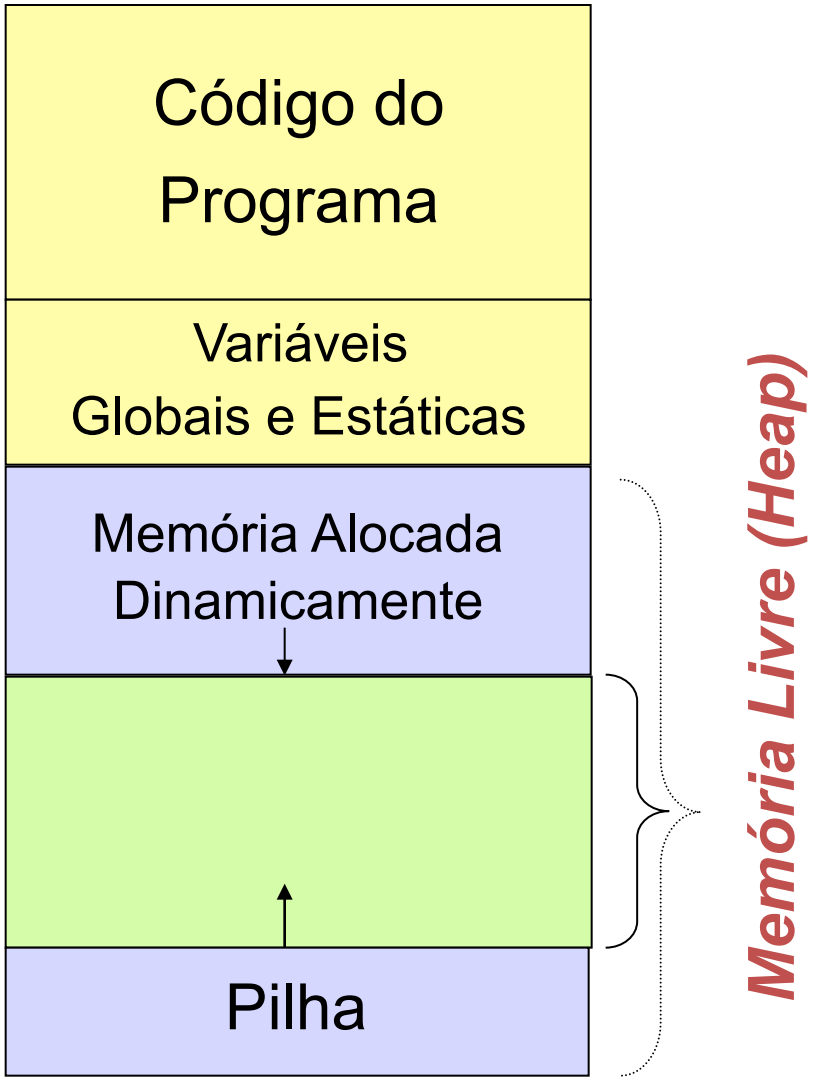
- O espaço alocado dinamicamente permanece reservado até que explicitamente seja liberado pelo programa.
  - Por isso, podemos alocar dinamicamente um espaço de memória numa função e acessá-lo em outra.

- A partir do momento que liberarmos o espaço, ele estará disponibilizado para outros usos e não podemos mais acessá-lo.
  - Se o programa não liberar um espaço alocado, este será automaticamente liberado quando a execução do programa terminar.

Alocação da Memória Principal

- Para executar um determinado programa, o S.O. carrega na memória o código do programa, em linguagem de máquina. Além disso, o S.O. reserva os espaços necessários para armazenar as variáveis globais (e estáticas) do programa.

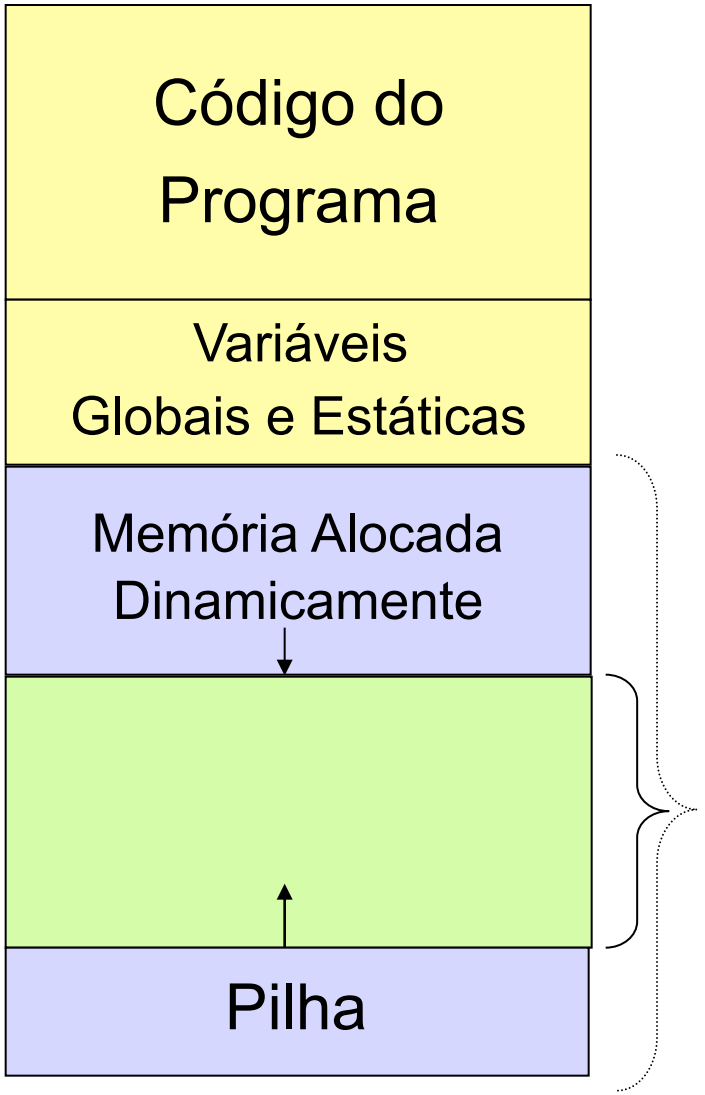
- O restante da memória livre é utilizado pelas variáveis locais e pelas variáveis alocadas dinamicamente.



Alocação da Memória Principal

- Cada vez que uma função é chamada, o S.O. reserva o espaço necessário para as variáveis locais da função. Este espaço pertence à pilha de execução e, quando a função termina, é liberado.

- A memória não ocupada pela pilha de execução pode ser requisitada dinamicamente. Se a pilha tentar crescer mais do que o espaço disponível existente, dizemos que ela "estourou" e o programa é abortado com erro.



Alocação Dinâmica de Memória

- As funções *calloc, malloc e realloc* permitem alocar blocos de memória em tempo de execução.

- Protótipo da função *malloc*:

void * malloc(**size_t** n);

/ retorna um ponteiro void para n bytes de memória não iniciados. Se não há memória disponível malloc retorna NULL */*

Funções para Alocar e Liberar memória

- A função *malloc* é usada para alocar espaço para armazenarmos valores de qualquer tipo. Por este motivo, *malloc* retorna um ponteiro genérico, para um tipo qualquer, representado por **void*** , que pode ser convertido automaticamente pela linguagem para o tipo apropriado na atribuição.

- No entanto, é comum fazermos a conversão explicitamente, utilizando o operador de molde de tipo (cast).

- Então:

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

Funções para Alocar e Liberar memória

- Se não houver espaço livre suficiente para realizar a alocação, a função retorna um endereço nulo (representado pelo símbolo NULL, definido em *stdlib.h*).

- Podemos tratar o erro na alocação do programa simplesmente verificando o valor de retorno da função *malloc*

- Ex: imprimindo mensagem e abortando o programa com a função *exit*, também definida na *stdlib*.

```
v = (int*) malloc(10*sizeof(int));
if (v == NULL) {
    printf("Memoria insuficiente.\n");
    exit(1);    /* aborta o programa e retorna 1 para o sist. operacional */
} ...
```

Alocação da Memória Principal

Exemplos:

- Código que aloca 1000 bytes de memória livre:

```
char *p;
p = malloc(1000);
```

- Código que aloca espaço para 50 inteiros:

```
int *p;
p = malloc(50*sizeof(int));
```

  - Obs.: O operador *sizeof*() retorna o número de bytes de um determinado tipo de variável.

Alocação Dinâmica de Memória

- As funções *calloc* e *malloc* permitem alocar blocos de memória em tempo de execução.

- Protótipo da função *calloc*:

void * calloc(**size_t** n, **size_t** size);

/ calloc retorna um ponteiro para um array com n elementos de tamanho size cada um ou NULL se não houver memória disponível. Os elementos são iniciados em zero */*

Alocação Dinâmica de Memória

- O ponteiro retornado por tanto pela função *malloc* quanto pela *calloc* devem ser convertido para o tipo de ponteiro que invoca a função

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

int *ai = (int *) calloc (n, sizeof(int));
/* aloca espaço para um array de n inteiros */
```

- toda memória não mais utilizada deve ser liberada através da função *free()*:

```
free(ai);  /* libera todo o array */
free(pi);
```

Funções para Alocar e Liberar memória

- A função **realloc()** serve para realocar memória. A função modifica o tamanho da memória previamente alocada apontada por ***ptr** para aquele especificado por **num**. O valor de **num** pode ser maior ou menor que o original.

- Protótipo:

  void *realloc (void *ptr, unsigned int num);

Alocação Dinâmica

VETORES E MATRIZES

Vetores e alocação dinâmica

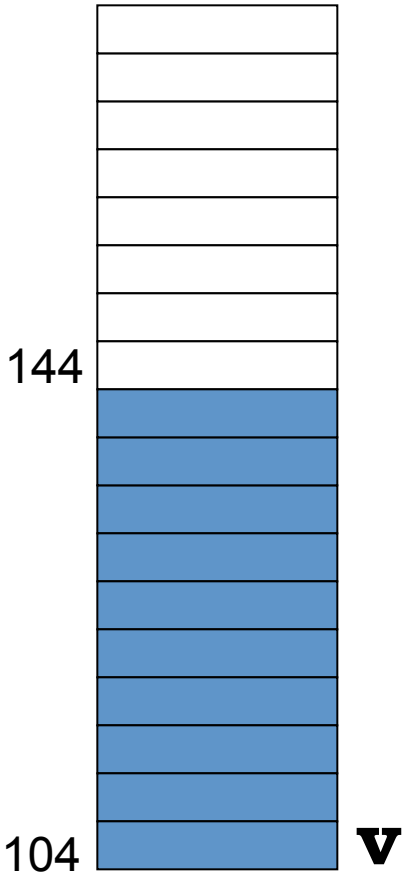
- A forma mais simples de estruturarmos um conjunto de dados é por meio de vetores.

- Definimos um vetor em C da seguinte forma:

int v[10];

- Esta declaração diz que:
  - v é um vetor de inteiros dimensionado com 10 elementos, isto é, reservamos um espaço de memória **contínuo** para armazenar 10 valores inteiros.
  - Assim, se cada **int** ocupa 4 bytes, a declaração reserva um espaço de memória de 40 bytes

Vetores e alocação dinâmica



- O acesso a cada elemento do vetor é feito através de uma indexação da variável v .
- Em C, a indexação de um vetor varia de zero a *n*-1, onde *n* representa a dimensão do vetor.
  - v[0] acessa o primeiro elemento de v
  - v[1] acessa o segundo elemento de v
  - ...
  - mas v[10] invade a memória

Vetores e alocação dinâmica

- Suponha que desejamos armazenar uma quantidade *n* de valores inteiros,porém essa quantidade somente é conhecida pelo usuário.

- Como proceder a reserva de memória necessária para os dados?
  - Se declararmos um vetor grande podemos incorrer em dois riscos:
    - ainda assim ser pequeno o bastante para não caber os dados
    - ou grande demais e desperdiçar memória
  - Solicitar a memória necessária assim que a quantidade for conhecida

Vetores e alocação dinâmica

```
#include <stdio.h>
#include <stdlib.h>
int main () {
      int *v, n, i;
      printf("Qual o tamanho do vetor que deseja:");
      scanf("%d", &n);
      v = (int *) calloc(n, sizeof(int));        /* aloca um vetor de n posições inteiras */
      for (i =0; i<n; i++) {
         printf("Informe o %dº elemento: ", i+1);
         scanf("%d", &v[i]);                     /* armazena o valor no vetor  na posição i */
       }
       for (i =0; i<n; i++)
          printf("%d ", v[i]);
      free(v);                                   /* libera a memória alocada para o vetor */
      return 0;
}
```

Alocação dinâmica de matrizes

- A alocação dinâmica de memória para matrizes é realizada da mesma forma que para vetores, com a diferença que teremos um ponteiro apontando para outro ponteiro que aponta para o valor final, o que é denominado indireção múltipla.
  - A indireção múltipla pode ser levada a qualquer dimensão desejada.

Alocação dinâmica de matrizes

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

float **Alocar_matriz_real(int m, int n) {
    float **v;                      /* ponteiro para a matriz */
    int i;                          /* variável auxiliar */

    if (m < 1 || n < 1) {           /* verifica parâmetros */
      printf ("** Erro: Parametro invalido **\n");
      return NULL;
    }
```

Alocação dinâmica de matrizes

```
    v = (float **) calloc(m, sizeof(float *));      /* aloca as linhas da matriz */
    if (v == NULL) {
       printf ("** Erro: Memoria Insuficiente **");
       return NULL;
    }
    for ( i = 0; i < m; i++ ) {                     /* aloca as colunas da matriz */
       v[i] = (float*) calloc(n, sizeof(float));
       if (v[i] == NULL) {
          printf ("** Erro: Memoria Insuficiente **");
          return NULL;
       }
    }
    return v;                                       /* retorna o ponteiro para a matriz */
}
```

Alocação dinâmica de matrizes

```
float ** liberar_matriz_real (int m, int n, float **v) {
    int i;                                          /* variável auxiliar */

    if (v == NULL) return NULL;

    if (m < 1 || n < 1) {                  /* verifica parâmetros recebidos */
       printf ("** Erro: Parametro invalido **\n");
       return v;
    }
    for (i = 0; i < m; i++) free (v[i]);   /* libera as linhas da matriz */

    free (v);                                       /* libera a matriz */
    return NULL;                           /* retorna um ponteiro nulo */
}
```

Alocação dinâmica de matrizes

```
void main () {
    float **mat; /* matriz a ser alocada */
    int l, c;                    /* numero de linhas e colunas */
    ...
    /* outros comandos, inclusive inicialização de l e c */
    mat = Alocar_matriz_real (l, c);
     /* outros comandos utilizando mat[][] normalmente */
    if (Liberar_matriz_real (l, c, mat) != NULL) {
       printf("Erro ao desalocar a matriz!\n");
       exit(1);
    }
    ...
}
```

Exercícios

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.

Exercícios

2) Crie uma função que aloca e lê um vetor de n inteiros. Crie outra função que recebe o vetor e retorna o maior e o menor valor.

   O programa principal deve executar as duas funções, imprimir os valores retornados pela 2ª função e liberar a memória alocada pela 1ª função.

Exercícios

3) Faça um programa que simule virtualmente a memória de um computador: o usuário começa especificando o tamanho da memória (define quantos bytes tem a memória), e depois ele irá ter 2 opções: inserir um dado em um determinado endereço, ou consultar o dado contido em um determinado endereço. A memória deve iniciar com todos os dados zerados.

Exercícios

3) Desenvolva um programa que calcule a soma de duas matrizes MxN de números reais (double). A implementação deste programa deve considerar as dimensões fornecida pelo usuário (Dica: represente a matriz através de variáveis do tipo *double* **, usando alocação dinâmica de memória)..