

Estrutura de Dados

Alocação Dinâmica

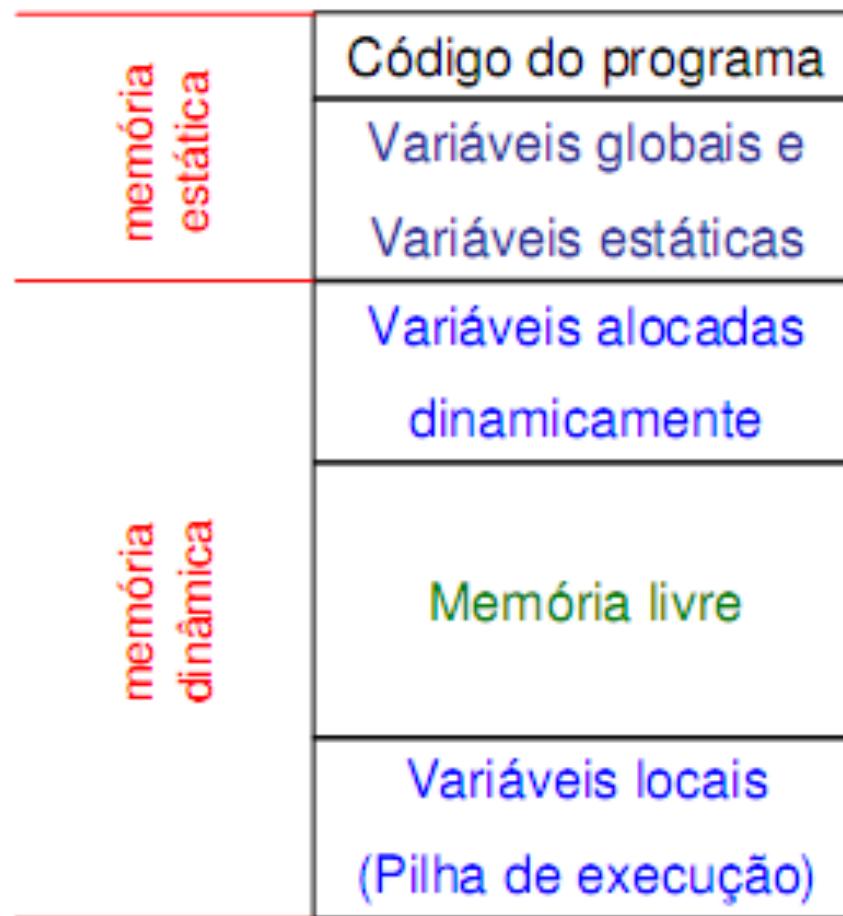
Uso da memória

- Uso de variáveis globais (e estáticas):
 - Espaço reservado para uma variável global existe enquanto o programa estiver sendo executado.
- Uso de variáveis locais:
 - Espaço existe apenas enquanto a função que declarou a variável está sendo executada;
 - Liberado para outros usos quando a execução da função termina.
- Variáveis globais ou locais podem ser simples ou vetores:
 - Para vetor, é necessário informar o número máximo de elementos;
 - pois o compilador precisa calcular o espaço a ser reservado.

Uso da memória

- Alocação dinâmica:
 - Espaço de memória é requisitada em tempo de execução;
 - Espaço permanece reservado até que seja explicitamente liberado:
 - Depois de liberado, espaço estará disponibilizado para outros usos e não pode mais ser acessado.
 - Espaço alocado e não liberado explicitamente, será automaticamente liberado quando ao final da execução.

Uso da memória



Funções: *stdlib.h*

- Contém uma série de funções pré-definidas:
 - Funções para tratar alocação dinâmica de memória;
 - constantes pré-definidas.
- Função “`sizeof`”:
 - retorna o número de bytes ocupado por um tipo
- Função “`malloc`”:
 - Recebe como parâmetro o número de bytes que se deseja alocar
 - Retorna um ponteiro genérico para o endereço inicial da área de memória alocada, se houver espaço livre:
 - Ponteiro genérico é representado por `void*`;
 - Ponteiro é convertido automaticamente para o tipo apropriado;
 - Ponteiro pode ser convertido explicitamente.
 - Retorna um endereço nulo, se não houver espaço livre:
 - Representado pelo símbolo `NULL`.

Exemplo

- Alocação dinâmica de um vetor de inteiros com 10 elementos
 - malloc retorna o endereço da área alocada para armazenar valores inteiros;
 - ponteiro de inteiro recebe endereço inicial do espaço alocado

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

Tratamento de erro: *malloc*

- Imprime mensagem de erro;
- Aborta o programa (com a função exit);

```
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 */
}
```

Funções: *stdlib.h*

- Função “*free*”:
 - Recebe como parâmetro o ponteiro da memória a ser liberada
 - A função *free* deve receber um endereço de memória que tenha sido alocado dinamicamente

free (v);

Exemplo

```
/* Cálculo da média e da variância de n reais */

#include <stdio.h>
#include <stdlib.h>
...
int main ( void )
{
    int i, n;
    float *v;
    float med, var;

    /* leitura do número de valores */
    scanf("%d", &n);
    /* alocação dinâmica */
    v = (float*) malloc(n*sizeof(float));
    if (v==NULL) {
        printf("Memoria insuficiente.\n");
        return 1;
    }
```

```
/* leitura dos valores */
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);
/* libera memória */
free(v);
return 0;
}
```

Vetores Locais a Funções

```
float* prod_vetorial (float* u, float* v)
{
    float p[3];
    p[0] = u[1]*v[2] - v[1]*u[2];
    p[1] = u[2]*v[0] - v[2]*u[0];
    p[2] = u[0]*v[1] - v[0]*u[1];
    return p; /* ERRO: não podemos retornar endereço de área local (p é vetor) */
}
```

- Variável **p** declarada localmente:
 - Área de memória que a variável **p** ocupa deixa de ser válida quando a função **prod_vetorial** termina;
 - Função que chama **prod_vetorial** não pode acessar a área apontada pelo valor retornado.

Vetores Locais a Funções

```
float* prod_vetorial (float* u, float* v)
{
    float *p = (float*) malloc(3*sizeof(float));
    p[0] = u[1]*v[2] - v[1]*u[2];
    p[1] = u[2]*v[0] - v[2]*u[0];
    p[2] = u[0]*v[1] - v[0]*u[1];
    return p;
}
```

- Variável **p** alocada dinamicamente:
 - Área de memória que a variável **p** ocupa permanece válida mesmo após o término da função **prod_vetorial**;
- Função que chama **prod_vetorial** pode acessar o ponteiro retornado;
- Problema - alocação dinâmica para cada chamada da função:
 - Ineficiente do ponto de vista computacional;
 - Requer que a função que chama seja responsável pela liberação do espaço.

Vetores Locais a Funções

```
void prod_vetorial (float* u, float* v, float* p)
{
    p[0] = u[1]*v[2] - v[1]*u[2];
    p[1] = u[2]*v[0] - v[2]*u[0];
    p[2] = u[0]*v[1] - v[0]*u[1];
}
```

- Espaço de memória para o resultado passado pela função que chama:
 - Função **prod_vetorial** recebe três vetores:
 - dois vetores com dados de entrada
 - um vetor para armazenar o resultado
- Solução mais adequada pois não envolve alocação dinâmica.