Estrutura de Dados

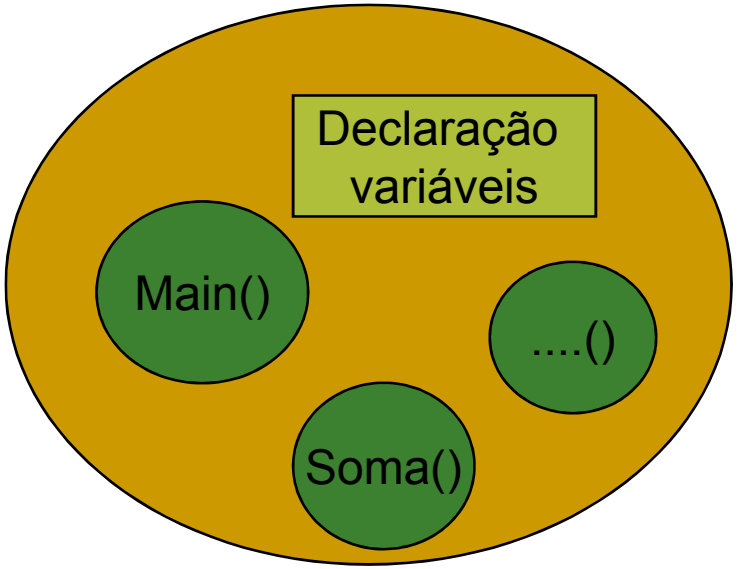
Alocação Dinâmica de Memória

Profa.: Márcia Sampaio Lima

EST - UEA

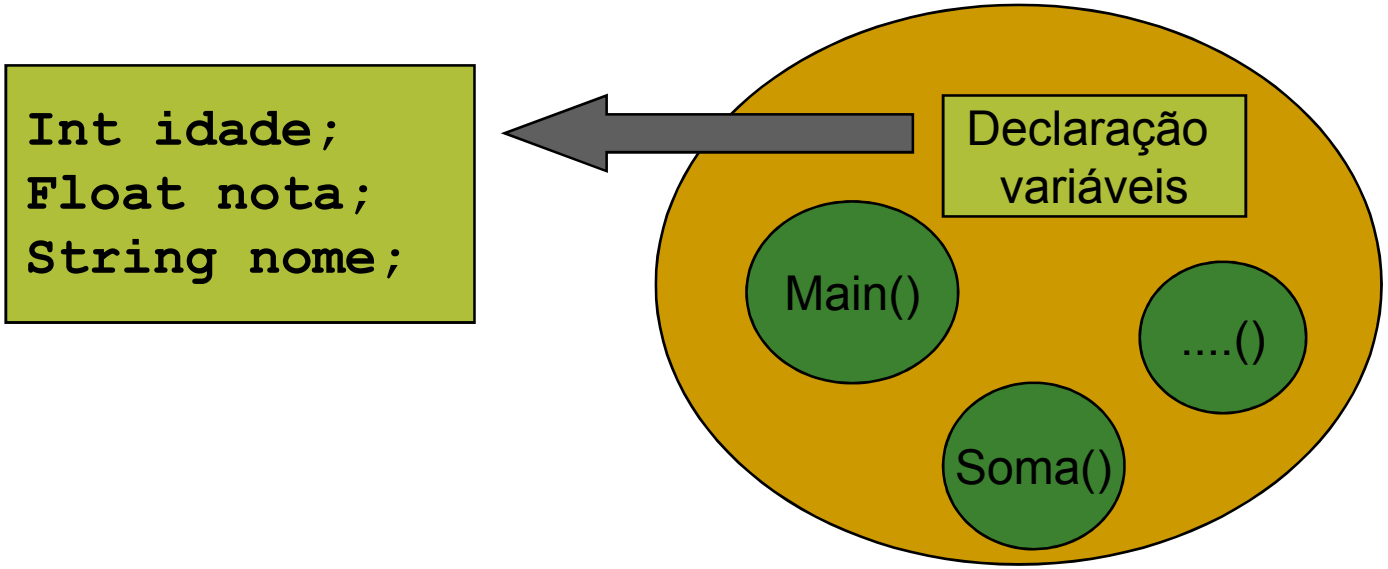
Uso da memória

- Variáveis globais (e estáticas):
  - Espaço reservado para uma variável global existe enquanto o programa estiver sendo executado.



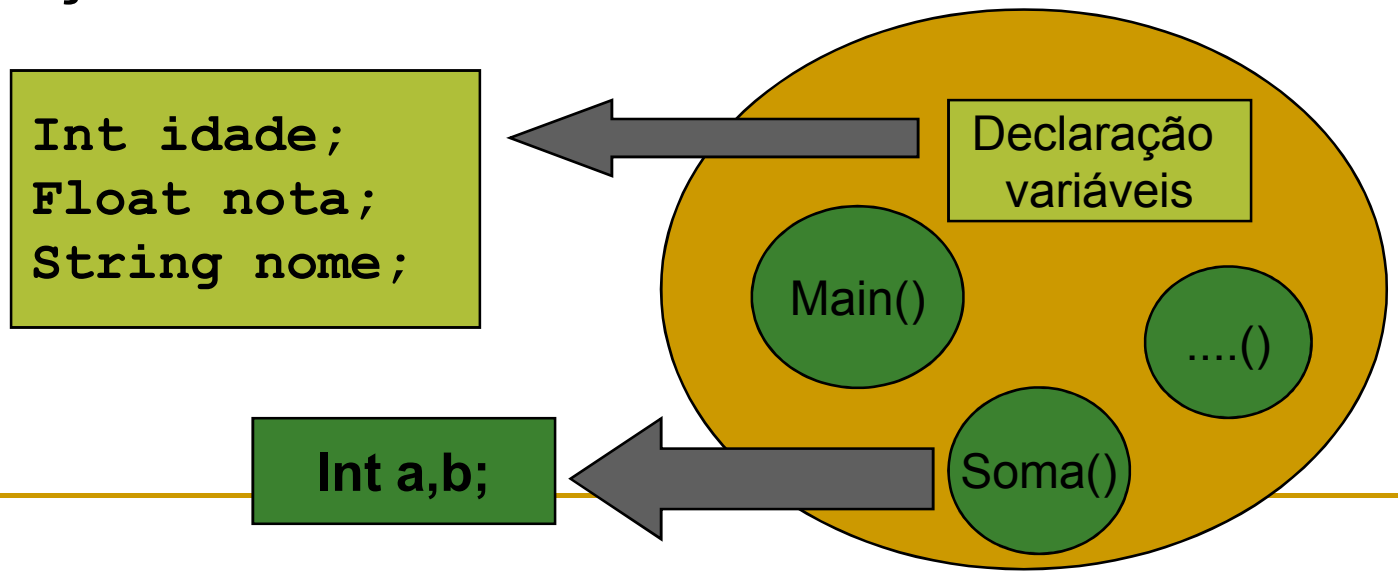
Uso da memória

- Variáveis globais (e estáticas):
  - Espaço reservado para uma variável global existe enquanto o programa estiver sendo executado.



Uso da memória

- 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

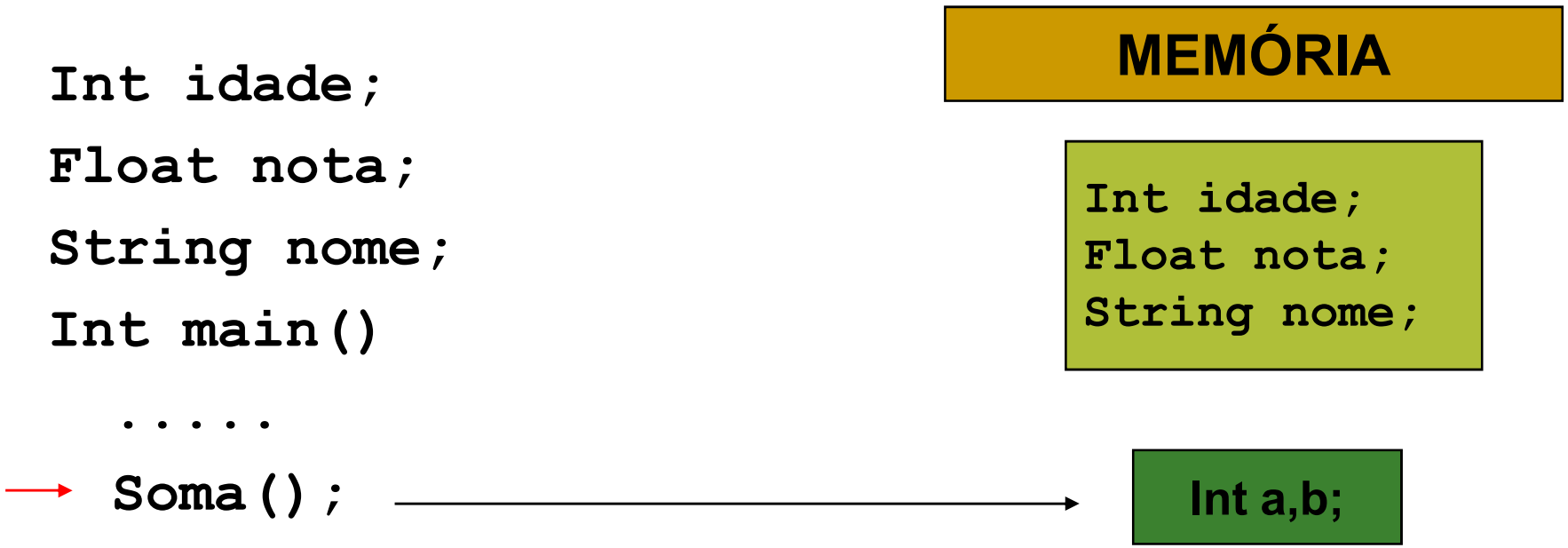


```
Int idade;
Float nota;
String nome;

Int Main(){
      .....


}
```

MEMÓRIA

```
Int idade;
Float nota;
String nome;
```

```
Int idade;
Float nota;
String nome;
Int main()
   .....
   Soma();

}
```



```
Int idade;
Float nota;
String nome;
Int main()
   .....
   Soma();
   cout << idade;

}
```

MEMÓRIA

```
Int idade;
Float nota;
String nome;
```

```
Main(){
   Int idade;
   Float nota;
   String nome;
   .....
   Soma();
   cout << idade;

}
```

MEMÓRIA

Uso da memória

- 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

- As declarações abaixo alocam memória para diversas variáveis.
- A alocação é ***estática***, pois acontece antes que o programa comece a ser executado:

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

Uso da memória

- Alocação dinâmica:
  - 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.
  - Espaço de memória é requisitado em tempo de execução.

Uso da Memória

- Alocação dinâmica:
  - 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 ao final da execução.

Uso da memória

- Memória estática:
  - código do programa
  - variáveis globais
  - variáveis estáticas
- Memória dinâmica:
  - variáveis alocadas dinamicamente
  - memória livre
  - variáveis locais

Uso da memória

- Alocação dinâmica de memória:
  - usa a memória livre
  - se o espaço de memória livre for menor que o espaço requisitado, a alocação não é feita e o programa pode prever tratamento de erro.

Uso da memória

- pilha de execução:
  - utilizada para alocar memória quando ocorre chamada de função:
    - sistema reserva o espaço para as variáveis locais da função.
    - quando a função termina, espaço é liberado (desempilhado)
    - se a pilha tentar crescer mais do que o espaço disponível existente, programa é abortado com erro

Uso da memória

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

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

- Funções da biblioteca padrão “stdlib.h”
  - Funções para tratar alocação dinâmica de memória: alocar e liberar espaço.
  - Constantes pré-definidas

Uso da memória

- **Função malloc()**
  - A função malloc (*memory allocation*) aloca um bloco de bytes consecutivos na memória do computador e devolve o endereço desse bloco.
  - 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 deve ser convertido para o tipo apropriado
  - Retorna nulo, se não houver espaço livre: NULL

Uso da memória

- **Função malloc**
  - No seguinte fragmento de código, malloc() aloca 1 byte:

```
char *ptr;
ptr = (char*) malloc(1);
scanf("%c",ptr);
```

Uso da memória

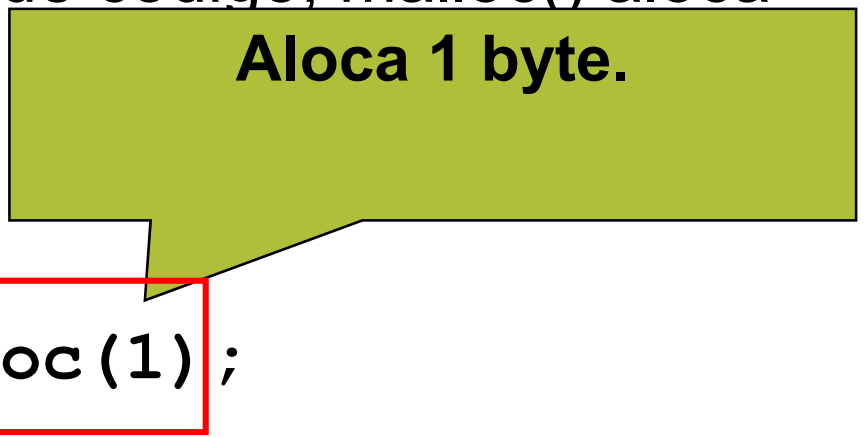
- **Função malloc**
  - No seguinte fragmento de código, malloc() aloca 1 byte:

Cria um pointer para um tipo char.

```
char *ptr;
ptr = (char*) malloc(1);
scanf("%c",ptr);
```

Uso da memória

- **Função malloc**
  - No seguinte fragmento de código, malloc() aloca 1 byte:




```
char *ptr;
sptr = (char*) malloc(1);
scanf("%c",ptr);
```

Uso da memória

- **Função malloc**
  - No seguinte fragmento de código, malloc() aloca 1 byte:

Converte para o tipo apropriado. Char*

```
char *ptr;
ptr = (char*) malloc(1);
scanf("%c",ptr);
```

Uso da memória

- **Função malloc**
  - No seguinte fragmento de código, malloc() aloca 1 byte:

```
char *ptr;
ptr = (char*) malloc(1);
scanf("%c",ptr);
```

Leitura

Uso da memória

- Codificar()

Uso da memória

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

- Foi declarado um ponteiro p do tipo int, e alocado um endereço de memória (4 bytes).
- A segunda referência ao tipo (int*) na instrução está relacionada a variável de retorno, ou seja, converte um ponteiro do tipo genérico para um do tipo int.
- A terceira referência ao tipo (int) na instrução está relacionada a alocação solicitada, ou seja, uma solicitação de endereço de tamanho int.

Uso da memória

- `float *p = (float*)(malloc(sizeof(float);`
  - Alocação para um tipo float

- `struct aluno *p;`
- `p = (struct aluno*)(malloc(sizeof(struct aluno);`

  - Alocação para um tipo struct aluno

Uso da memória

- Função “sizeof”:
  - Retorna o número de bytes ocupado por um tipo.

```
sizeof(int);

char *ptr;
sptr = (char*) malloc(sizeof(char));
scanf("%c",ptr);
```

Uso da memória

- Codificar()

Uso da memória

- Função “sizeof”:
  - Retorna o número de bytes ocupado por um tipo.

```
cout << "Tam. de um char: "<< sizeof(char);
cout << "Tam. de um int: "<< sizeof(int);
cout << "Tam. de um float:"<< sizeof(float);
```

Tabela: Valores para uma máquina com arquitetura de 32 bits

Tipo	Nº reservado de bytes	Faixa de valores
char	1	-128 a 127
unsigned char	1	0 a 255
signed char	1	-128 a 127
int	4	-2.147.483.648 a 2.147.483.647
unsigned int	4	0 a 4.294.967.295
signed int	4	-2.147.483.648 a 2.147.483.647
short int	2	-32.768 a 32.767
unsigned short int	2	0 a 65.535
signed short int	2	-32.768 a 32.767
long int	4	-2.147.483.648 a 2.147.483.647
signed long int	4	-2.147.483.648 a 2.147.483.647
unsigned long int	4	0 a 4.294.967.295
float	4	10-38 a 1038
double	8	10-308 a 10308
long double	10	10-4932 a 104932

Uso da memória

- Codificar()

Uso da memória

- 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.

```
char *ptr;
ptr = (char*) malloc(sizeof(char));
scanf("%c",ptr);
free(ptr);
```

Uso da memória

- Codificar()

Uso da memória

- Exemplo 2:
  - 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));
```

Uso da memória

- Exemplo 2:
  - Alocação dinâmica de um **vetor de inteiros com 10 elementos**.
    - Malloc() retorna o endereço da área alocada para

Cria um ponteiro para inteiro

dereço inicial do espaço

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

Uso da memória

- Exemplo 2:
  - Alocação dinâmica de um **vetor de inteiros com 10 elementos**.
    - Malloc() retorna o endereço da área alocada para armazenar valores inteiros.
    - Ponteiro de inteiro


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

Uso da memória

- Exemplo 2:
  - Alocação dinâmica de um **vetor de inteiros com 10 elementos**.
    - Malloc() retorna o endereço da área alocada para armazenar valores inteiros.
    - Ponteiro de inteiro para o espaço alocado.

Converte para o tipo apropriado

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

Uso da memória

- Exemplo 2:
  - 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 o endereço inicial do espaço alocado.

Ponteiro recebe endereço inicial do espaço alocado

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

Uso da memória

- v armazena endereço inicial de uma área contínua de memória suficiente para armazenar 10 valores inteiros.
- v pode ser tratado como um vetor declarado estaticamente
  - v aponta para o inicio da área alocada
  - v[0] acessa o espaço para o primeiro elemento
  - v[1] acessa o segundo
  - .... até v[9]

Uso da memória

- Codificar()

Uso da memória

- Tratamento de erro após chamada a malloc():
  - Ocorre qdo não há memória suficiente para ser alocada.
  - Imprime mensagem de erro
  - Aborta o programa (com a função exit)

Uso da memória

Tenta alocar!!!

- Tratamento de erro após chamada à malloc():

```
…
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 */
}
…
free(v);
```

Uso da memória

(tipo*)(malloc(quantidade_de_memoria*sizeof(tipo)));

- Tratamento após chamada a malloc():

```
…
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 */
}
…
free(v);
```

Uso da memória

- Tratamento de erro ( ):

```
…
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 */
}
…
free(v);
```

Verifica se alocou com sucesso ou não. Se houver erro então v ==NULL

Uso da memória

- Tratamento de erro após chamada a malloc():

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

Imprime a mensagem de erro, e Aborta o programa

Uso da memória

- Tratamento de erro após chamada a malloc():

```
…
v = (int*) malloc(10*sizeof(int));
if (v==NULL)
{
 printf("Memoria
 exit(1); /* abor                    a 1
  para o sist. op
}
…
free(v);
```

Se NÃO houver erro continua a execução normal do programa...

Uso da memória

- Codificar()

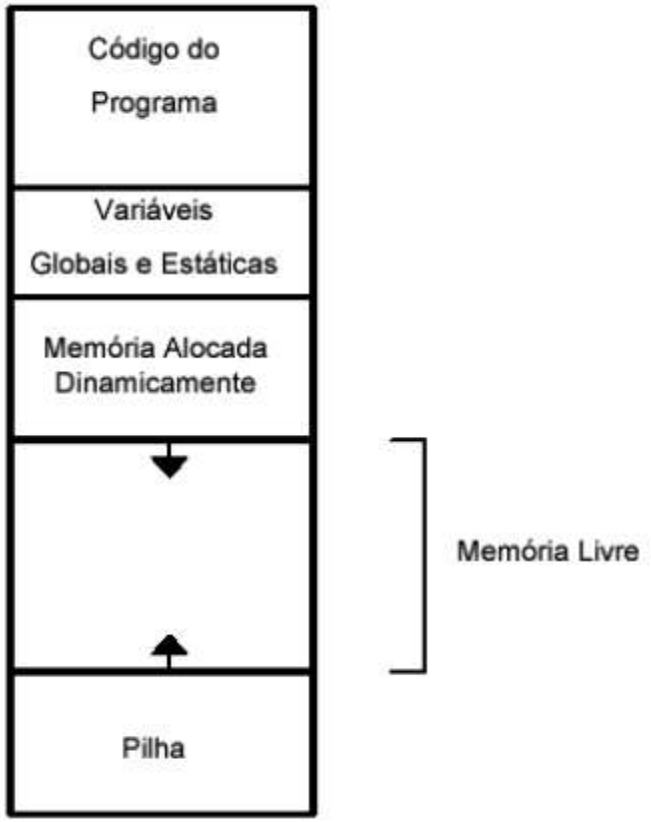
```c
#include <stdio.h>
#include <stdlib.h>

int main (){
   int *p;
   p=(int *)malloc(sizeof(int));
   if (!p)
      { printf ("** Erro: Memoria Insuficiente **");
        Exit(1);
      }
   else{
       printf ("** Memoria Alocada com Sucesso **");
        }
   return (0);
}
```

Uso da memória

- Codificar()

Uso de Memória

```
void * malloc(int num_bytes);

void free(void * p);
```



Exercício

- Faça um programa em C que leia um número inteiro **num** e aloque espaço suficiente para **num** números **float.**
  - Vetor de num números do tipo float.
- Leia tais números.
- Calcule e mostre a média desses números.

```
main(){
   int num,i;
   float media=0, *v;

   cout <<"Informe o valor num: ";
   cin >> num;

   v = (float*) malloc(num*sizeof(float));

   if (!v){
        cout << "Memoria insuficiente!!" << endl;
        exit(1);
   }else{cout << "Memoria alocada com sucesso!!" << endl;
        }

   for (i=0;i<num;i++){
        cout <<"Informe o valor de v[ "<< i <<" ]: ";
        cin >> v[i];
   }

   for (i=0;i<num;i++){
        media += v[i];
   }
   cout <<"Media = " << (media/num) << endl;
   free(v);
}
```

Alocação Dinâmica

- Função “realloc()”:
  - Essa função faz um bloco já alocado crescer ou diminuir, **preservando o conteúdo já existente**.

(tipo*) realloc(tipo *apontador, int novo_tamanho)

```
int *x, i;
x = (int *) malloc(4000*sizeof(int));
for(i=0;i<4000;i++)
 v[i] = rand()%100;
x = (int *) realloc(x, 8000*sizeof(int));
x = (int *) realloc(x, 2000*sizeof(int));
free(x);
```

Alocação Dinâmica

- Codificar...