



# Alocação Dinâmica

# Definição

- Sempre que escrevemos um programa, é preciso reservar espaço para as informações que serão processadas.
- Para isso utilizamos as variáveis
  - Uma variável é uma posição de memória que armazena uma informação que pode ser modificada pelo programa.
  - Ela deve ser definida antes de ser usada.

# Esquema de um Programa na Memória

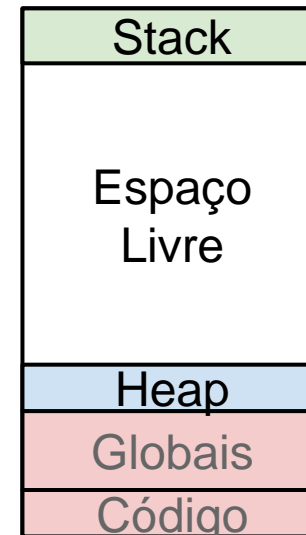
- Podemos simplificar um programa com uma esquema como o abaixo

-  [Sempre] Código  
 Que ocupa espaço

- 🔗 [Pode ter] Variáveis globais  
🌸 Que ocupa espaço

- 🔗 [Sempre] Stack ou Pilha
  - 🌸 Utilizado pelo hardware para chamar funções

- 🔗 [Sempre] Heap
  - ✿ Alocação dinâmica (esta aula)



# Definição

- Infelizmente, nem sempre é possível saber o quanto de memória um programa irá precisar.
- Surge então a necessidade de se utilizar ponteiros
  - Ponteiro é uma variável que guarda um endereço de memória.

# Definição

- A *alocação dinâmica* permite ao programador criar “variáveis” em tempo de execução, ou seja, alocar memória para novas variáveis quando o programa está sendo executado, e não apenas quando se está escrevendo o programa.

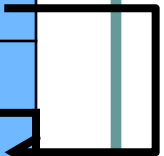
# Alocando memória

Memória		
#	var	conteúdo
119		
120		
121	int *n	NULL
122		
123		
124		
125		
126		
127		
128		
129		

Alocando 5  
posições de  
memória em int \* n



Memória		
#	var	conteúdo
119		
120		
121	int *n	#123
122		
123	n[0]	11
124	n[1]	25
125	n[2]	32
126	n[3]	44
127	n[4]	52
128		
129		



# Alocação Dinâmica

- A linguagem C ANSI usa apenas 4 funções para o sistema de alocação dinâmica, disponíveis na `stdlib.h`:
  - `malloc`
  - `calloc`
  - `realloc`
  - `free`

# Alocação Dinâmica - malloc

- malloc

- A função malloc() serve para alocar memória e tem o seguinte protótipo:

**void \*malloc (unsigned int num);**

- Funcionalidade

- Dado o número de bytes que queremos alocar (**num**), ela aloca na memória e retorna um ponteiro **void\*** para o primeiro byte alocado.



# Alocação Dinâmica - malloc

- O ponteiro **void\*** pode ser atribuído a qualquer tipo de ponteiro via *type cast*. Se não houver memória suficiente para alocar a memória requisitada a função malloc() retorna um ponteiro nulo.

# Alocação Dinâmica - malloc

- Alocar 1000 bytes de memória livre.

```
char *p;
```

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

- Alocar espaço para 50 inteiros:

```
int *p;
```

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

# Alocação Dinâmica - malloc

- Operador **sizeof()**

- Retorna o número de bytes de um dado tipo de dado. Ex.: int, float, char...

- No exemplo anterior,

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

- **sizeof(int)** retorna 4 (número de bytes do tipo **int** na memória). Portanto, são alocados 200 bytes.

# Alocação Dinâmica - malloc

- Se não houver memória suficiente para alocar a memória requisitada, a função malloc() retorna um ponteiro nulo

```
01  #include <stdio.h>
02  #include <stdlib.h>
03  int main(){
04      int *p;
05      p = (int *) malloc(5*sizeof(int));
06      if(p == NULL){
07          printf("Erro: Memoria Insuficiente!\n");
08          system("pause");
09          exit(1);
10      }
11      int i;
12      for (i=0; i<5; i++){
13          printf("Digite o valor da posicao %d: ",i);
14          scanf("%d",&p[i]);
15      }
16      system("pause");
17      return 0;
18  }
```

# Alocação Dinâmica - calloc

- calloc

- A função calloc() também serve para alocar memória, mas possui um protótipo um pouco diferente:

**void \*calloc (unsigned int num, unsigned int size);**

- Funcionalidade

- Basicamente, a função calloc() faz o mesmo que a função malloc(). A diferença é que agora passamos a quantidade de posições a serem alocadas e o tamanho do tipo de dado alocado como parâmetros distintos da função.
- Atribui zero para ao valor do endereço de memória alocado

# Alocação Dinâmica - calloc

```
01  #include <stdio.h>
02  #include <stdlib.h>
03  int main(){
04      //alocação com malloc
05      int *p;
06      p = (int *) malloc(50*sizeof(int));
07      if(p == NULL){
08          printf("Erro: Memoria Insuficiente!\n");
09      }
10      //alocação com calloc
11      int *p1;
12      p1 = (int *) calloc(50,sizeof(int));
13      if(p1 == NULL){
14          printf("Erro: Memoria Insuficiente!\n");
15      }
16      system("pause");
17      return 0;
18  }
```

# Alocação Dinâmica - realloc

- realloc

- A função realloc() serve para realocar memória e tem o seguinte protótipo:

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

- Funcionalidade

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

# Alocação Dinâmica - realloc

- realloc

- Um ponteiro para o bloco é devolvido porque realloc() pode precisar mover o bloco para aumentar seu tamanho.
- Se isso ocorrer, o conteúdo do bloco antigo é copiado para o novo bloco, e nenhuma informação é perdida.

```
01  #include <stdio.h>
02  #include <stdlib.h>
03  int main(){
04      int i;
05      int *p = malloc(5*sizeof(int));
06      for (i = 0; i < 5; i++){
07          p[i] = i+1;
08      }
09      for (i = 0; i < 5; i++){
10          printf("%d\n",p[i]);
11      }
12      printf("\n");
13      //Diminui o tamanho do array
14      p = realloc(p,3*sizeof(int));
15      for (i = 0; i < 3; i++){
16          printf("%d\n",p[i]);
17      }
18      printf("\n");
19      //Aumenta o tamanho do array
20      p = realloc(p,10*sizeof(int));
21      for (i = 0; i < 10; i++){
22          printf("%d\n",p[i]);
23      }
24      system("pause");
25      return 0;
26  }
```



# Alocação Dinâmica - realloc

- Observações sobre realloc()
  - Se **\*ptr** for nulo, aloca **num** bytes e devolve um ponteiro (igual malloc);
  - se **num** é zero, a memória apontada por **\*ptr** é liberada (igual free).
  - Se não houver memória suficiente para a alocação, um ponteiro nulo é devolvido e o bloco original é deixado inalterado.

# Alocação Dinâmica - free

- free
  - Diferente das variáveis definidas durante a escrita do programa, as variáveis alocadas dinamicamente não são liberadas automaticamente pelo programa.
  - Quando alocamos memória dinamicamente é necessário que nós a liberemos quando ela não for mais necessária.
  - Para isto existe a função `free()` cujo protótipo é:  
**`void free (void *p);`**

# Alocação Dinâmica - free

- Assim, para liberar a memória, basta passar como parâmetro para a função `free()` o ponteiro que aponta para o início da memória a ser desalocada.
- Como o programa sabe quantos bytes devem ser liberados?
  - Quando se aloca a memória, o programa guarda o número de bytes alocados numa "tabela de alocação" interna.

# Alocação Dinâmica - free

```
01  #include <stdio.h>
02  #include <stdlib.h>
03  int main(){
04      int *p,i;
05      p = (int *) malloc(50*sizeof(int));
06      if(p == NULL){
07          printf("Erro: Memoria Insuficiente!\n");
08          system("pause");
09          exit(1);
10      }
11      for (i = 0; i < 50; i++){
12          p[i] = i+1;
13      }
14      for (i = 0; i < 50; i++){
15          printf("%d\n",p[i]);
16      }
17      //libera a memória alocada
18      free(p);
19      system("pause");
20      return 0;
21  }
```

# Alocação de memória de matrizes

```
1  #include <stdio.h>
2  #include <stdlib.h>
3
4  int **AlocaMatriz(int m, int n){
5      int **M;
6      int i;
7
8      M = (int **)malloc(sizeof(int *)*m);
9      if(M == NULL){
10         printf("Memoria insuficiente.\n");
11         exit(1);
12     }
13     for(i = 0; i < m; i++){
14         M[i] = (int *)malloc(sizeof(int)*n);
15         if(M[i] == NULL){
16             printf("Memoria insuficiente.\n");
17             exit(1);
18         }
19     }
20     return M;
21 }
```


# Liberar memória de matrizes

```
1 void LiberaMatriz(int **M, int m){  
2     int i;  
3     for(i = 0; i < m; i++)  
4         free(M[i]);  
5     free(M);  
6 }
```

# Alocação de struct

```
1  #include <stdio.h>
2  #include <stdlib.h>
3
4  struct ST_DADOS {
5      char nome[40];
6      float salario;
7
8      /* estrutura dentro de uma estrutura */
9      struct nascimento {
10         int ano;
11         int mes;
12         int dia;
13     } dt_nascimento;
14 };
15
16 int main(void) {
17     /* ponteiro para a estrutura */
18     struct ST_DADOS * p;
19
20     /* alocação de memória para o ponteiro da estrutura */
21     p = (struct ST_DADOS *) malloc(sizeof(struct ST_DADOS));
22
23     /* Libera memoria alocada para o ponteiro de estrutura */
24     free(p);
25
26
27     return 0;
28 }
29
```

# Alocação Dinâmica - free



Save The whales  
Feed The Hungry  
Free The Malloc()s