ALOCAÇÃO DINÂMICA EM C

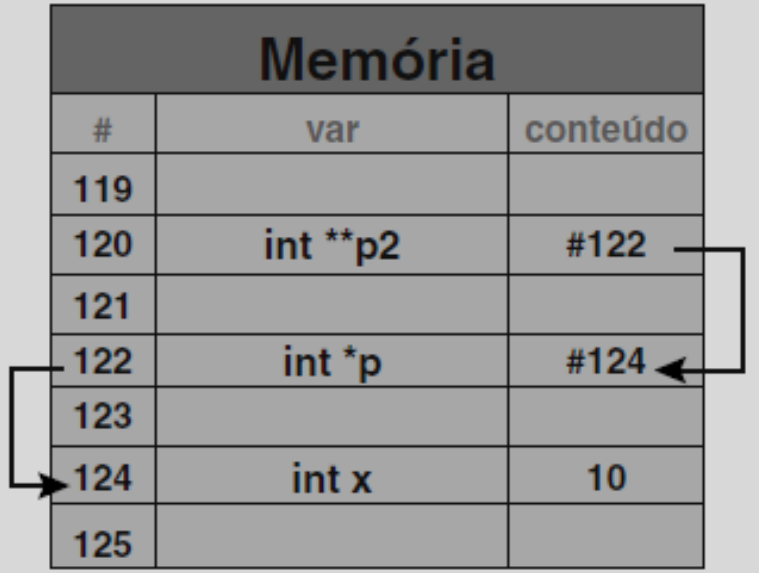
Prof. Manoel Pereira Junior
IFMG – Campus Formiga

- Ponteiros para ponteiros... Sim, é possível!

tipo_do_ponteiro **nome_do_ponteiro;

Exemplo: ponteiro para ponteiro

```
01  #include <stdio.h>
02  #include <stdlib.h>
03  int main(){
04    int x = 10;
05    int *p = &x;
06    int **p2 = &p;
07    //Endereço em p2
08    printf("Endereco em p2: %p\n",p2);
09    //Conteudo do endereço
10    printf("Conteudo em *p2: %p\n",*p2);
11    //Conteudo do endereço do endereço
12    printf("Conteudo em **p2: %d\n",**p2);
13    system("pause");
14    return 0;
15  }
```



```c
#include <stdio.h>
#include <stdlib.h>
int main(){
//variável inteira
int x;
//ponteiro para um inteiro (1 nível)
int *p1;
//ponteiro para ponteiro de inteiro (2 níveis)
int **p2;
//ponteiro para ponteiro para ponteiro de inteiro(3 níveis)
int ***p3;
return 0;
}
```

Prof. Manoel Pereira Junior
IFMG – Campus Formiga

```c
#include <stdio.h>
#include <stdlib.h>
int main(){
char letra='a';
char *ptrChar = &letra;
char **ptrPtrChar = &ptrChar;
char ***ptrPtr = &ptrPtrChar;
printf("Conteudo em *ptrChar: %c\n",*ptrChar);
printf("Conteudo em **ptrPtrChar: %c\n",**ptrPtrChar);
printf("Conteudo em ***ptrPtr: %c\n",***ptrPtr);
return 0;
}
```

```
Conteudo em *ptrChar: a
Conteudo em **ptrPtrChar: a
Conteudo em ***ptrPtr: a
```

Prof. Manoel Pereira Junior
IFMG – Campus Formiga

ALOCAÇÃO DINÂMICA...

- Toda variável deve ser declarada antes de ser usada!
- Nem sempre conhecemos a priori o tamanho das estruturas

Prof. Manoel Pereira Junior
IFMG – Campus Formiga

EXEMPLO

- Preciso processar os salários dos funcionários de uma empresa.

- Solução 1:

```c
#include <stdio.h>
#include <stdlib.h>
int main(){

float salarios[1000];

}
```

Essa solução é boa?

Prof. Manoel Pereira Junior
IFMG – Campus Formiga

- Ao declarar um array, necessariamente precisamos definir seu tamanho (quantidade de memória fixa)!
- Para contornar este problema, usamos arrays com ponteiros!
- Relembrando:

  (1) um array é um agrupamento sequencial de elementos na memória

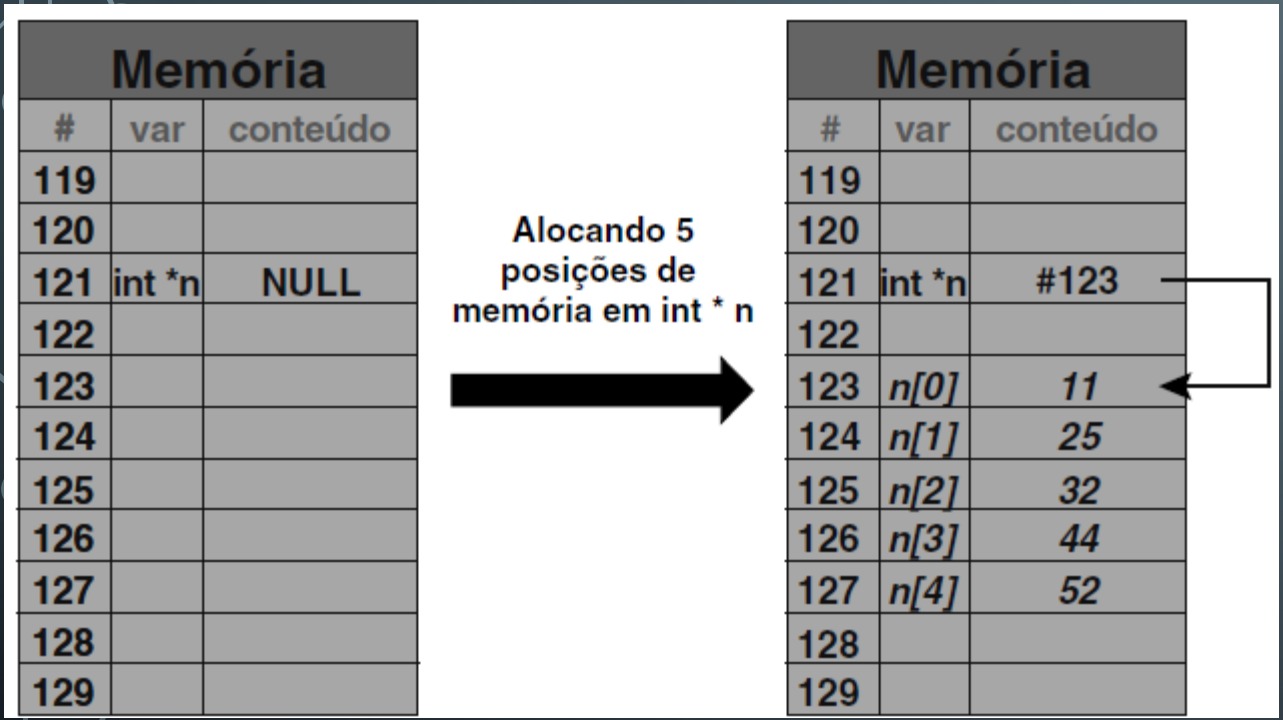
  (2) o nome de um array é um ponteiro para o primeiro elemento

Prof. Manoel Pereira Junior
IFMG – Campus Formiga

ALOCAÇÃO DINÂMICA É...

- Requisitar memória em tempo de execução!
- A alocação devolve um ponteiro (endereço) para o início do espaço alocado.

Prof. Manoel Pereira Junior
IFMG – Campus Formiga



(1) Ponteiro n apontando para **NULL**

(2) Requisitamos 5 posições de memória

(3) Recebemos as posições de #123 a #127

(4) n passa a se comportar como um array de 5 posições int(n)

Prof. Manoel Pereira Junior
IFMG – Campus Formiga

FUNÇÕES PARA ALOCAÇÃO...

- malloc
- calloc
- realloc
- free
- sizeof

Prof. Manoel Pereira Junior
IFMG – Campus Formiga

sizeof

- Usada para saber o tamanho (em bytes) de variáveis ou tipos.

```
sizeof nome_da_variável
sizeof (nome_do_tipo)
```

Prof. Manoel Pereira Junior
IFMG – Campus Formiga

sizeof

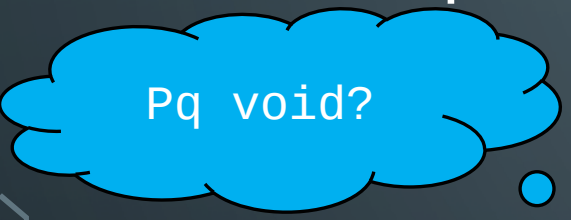
```c
#include <stdio.h>
#include <stdlib.h>
struct ponto{
int x,y;
};
int main(){
printf("Tamanho char: %d\n",sizeof(char));
printf("Tamanho int: %d\n",sizeof(int));
printf("Tamanho float: %d\n",sizeof(float));
printf("Tamanho double: %d\n",sizeof(double));
printf("Tamanho struct ponto: %d\n",sizeof(struct ponto));
int x;
double y;
printf("Tamanho da variavel x: %d\n",sizeof x);
printf("Tamanho da variavel y: %d\n",sizeof y);
return 0;
}
```

```
Tamanho char: 1
Tamanho int: 4
Tamanho float: 4
Tamanho double: 8
Tamanho struct ponto: 8
Tamanho da variavel x: 4
Tamanho da variavel y: 8
```

Prof. Manoel Pereira Junior
IFMG – Campus Formiga

malloc

- Usada para alocar memória durante a execução do programa.




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

num: o tamanho do espaço de memória a ser alocado

→Retorna **NULL** em caso de erro

Prof. Manoel Pereira Junior
IFMG – Campus Formiga

malloc

Essa operação de *"cast"* não é necessária para ponteiros!!! (embora possa ser feita)

```c
#include <stdio.h>
#include <stdlib.h>
int main(){
    int *p;
    p = (int *) malloc(5*sizeof(int));
    int i;
    for (i=0; i<5; i++){
        printf("Digite o valor da posicao %d: ",i);
        scanf("%d",&p[i]);
    }
    return 0;
}
```

```
Digite o valor da posicao 0: 1
Digite o valor da posicao 1: 2
Digite o valor da posicao 2: 3
Digite o valor da posicao 3: 4
Digite o valor da posicao 4: 5
```

Prof. Manoel Pereira Junior
IFMG – Campus Formiga

malloc

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

int main(){
    int *p;
    p = (int *) malloc(5*sizeof(int));
    if(p == NULL){
        printf("Erro: Memoria Insuficiente!\n");
        exit(1);
    }
    int i;
    for (i=0; i<5; i++){
        printf("Digite o valor da posicao %d: ",i);
        scanf("%d",&p[i]);
    }
    return 0;
}
```

Prof. Manoel Pereira Junior
IFMG – Campus Formiga

malloc – cuidado!

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

int main(){
    char *p;
    //aloca espaço para 1.000 chars
    p = malloc(1000);
    int *y;
    //aloca espaço para 250 inteiros
    y = malloc(1000);
    return 0;
}
```

Prof. Manoel Pereira Junior
IFMG – Campus Formiga

calloc

- Usada para alocar memória durante a execução do programa.

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

num: o número de elementos no array a ser alocado.

size: o tamanho de cada elemento do array.

→Retorna **NULL** em caso de erro

Prof. Manoel Pereira Junior
IFMG – Campus Formiga

calloc

```c
#include <stdio.h>
#include <stdlib.h>
int main(){
    //alocação com malloc
    int *p;
    p = malloc(50*sizeof(int));
    if(p == NULL){
        printf("Erro: Memoria Insuficiente!\n");
    }
    //alocação com calloc
    int *p1;
    p1 = calloc(50,sizeof(int));
    if(p1 == NULL){
        printf("Erro: Memoria Insuficiente!\n");
    }
    return 0;
}
```

Prof. Manoel Pereira Junior
IFMG – Campus Formiga

malloc x calloc

- Como vimos, as duas funções alocam memória dinamicamente. Mas qual a diferença entre elas?

- A função calloc, além de alocar a memória, inicializa todas as posições com “**zero**” e malloc não!

Prof. Manoel Pereira Junior
IFMG – Campus Formiga

realloc

- Usada para alocar memória ou realocar blocos de memória previamente alocados pelas funções malloc(), calloc() ou realloc().

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

***ptr**: ponteiro para um bloco de memória previamente alocado.

num: o tamanho em bytes do espaço de memória a ser alocado.

→Retorna **NULL** em caso de erro

Prof. Manoel Pereira Junior
IFMG – Campus Formiga

realloc

- Na prática, a função realloc() é utilizada para modificar o tamanho de estruturas em tempo de execução (diminuir ou aumentar o tamanho de uma estrutura).
- Quando faremos isso?

Prof. Manoel Pereira Junior
IFMG – Campus Formiga

realloc

```c
#include <stdio.h>
#include <stdlib.h>
int main(){
    int i;
    int *p = malloc(5*sizeof(int));
    for (i = 0; i < 5; i++){
        p[i] = i+1;
    }
    for (i = 0; i < 5; i++){
        printf("%d\n",p[i]);
    }
    printf("\n");
    //Diminui o tamanho do array
    p = realloc(p,3*sizeof(int));
    for (i = 0; i < 3; i++){
        printf("%d\n",p[i]);
    }
    printf("\n");
    //Aumenta o tamanho do array
    p = realloc(p,10*sizeof(int));
    for (i = 0; i < 10; i++){
        printf("%d\n",p[i]);
    }
    return 0;
}
```

```
1
2
3
4
5

1
2
3

1
2
3
4
5
0
135137
0
0
0
```

Prof. Manoel Pereira Junior
IFMG – Campus Formiga

free

- Diferentemente das variáveis declaradas durante o desenvolvimento do programa, as variáveis alocadas dinamicamente **não são** liberadas automaticamente por ele.
- Usada para desalocar memória previamente alocada.

Prof. Manoel Pereira Junior
IFMG – Campus Formiga

free

```
void free (void *p);
```

p: ponteiro para o início do bloco que será desalocado

Prof. Manoel Pereira Junior
IFMG – Campus Formiga

free

```c
#include <stdio.h>
#include <stdlib.h>
int main(){
    int *p,i;
    p = malloc(10*sizeof(int));
    if(p == NULL){
        printf("Erro: Memoria Insuficiente!\n");
        exit(1);
    }
    for (i = 0; i < 10; i++){
        p[i] = i+1;
    }
    for (i = 0; i < 10; i++){
        printf("%d\n",p[i]);
    }
    //libera a memória alocada
    free(p);
    return 0;
}
```

```
1
2
3
4
5
6
7
8
9
10
```

Prof. Manoel Pereira Junior
IFMG – Campus Formiga

free

```c
#include <stdio.h>
#include <stdlib.h>
int main(){
    int *p,i;
    p = malloc(10*sizeof(int));
    if(p == NULL){
        printf("Erro: Memoria Insuficiente!\n");
        exit(1);
    }
    for (i = 0; i < 10; i++){
        p[i] = i+1;
    }
    //libera a memória alocada
    free(p);
    //tenta imprimir o array
    //cuja memória foi liberada
    for (i = 0; i < 10; i++){
        printf("%d\n",p[i]);
    }
    return 0;
}
```

```
1
2
3
4
5
6
7
8
9
10
```

Por que consegui imprimir o vetor p?

Prof. Manoel Pereira Junior
IFMG – Campus Formiga

free

- Apenas libere a memória quando tiver certeza de que ela não será mais usada!
- não deixe ponteiros “soltos” (*dangling pointers*) – ataque de hackers!
- Sempre que usar free, atribua NULL para o ponteiro

Prof. Manoel Pereira Junior
IFMG – Campus Formiga

free

- Sempre que usar free, atribua NULL para o ponteiro

```c
#include <stdio.h>
#include <stdlib.h>
int main(){
    int *p;
    p = malloc(10*sizeof(int));
    free(p);
    p = NULL;
 }
```

Prof. Manoel Pereira Junior
IFMG – Campus Formiga

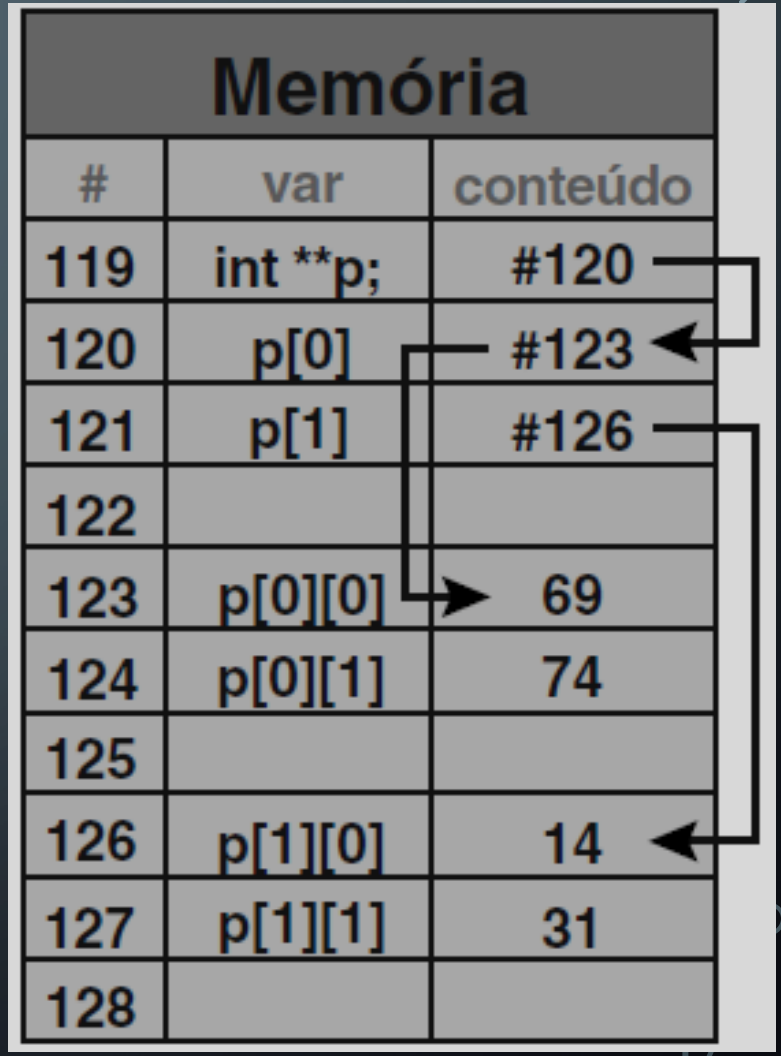
Alocando arrays multidimensionais...

- Usamos ponteiros para ponteiros
- O nível de apontamento depende da quantidade de dimensões que o array terá

Prof. Manoel Pereira Junior
IFMG – Campus Formiga

Alocando uma matriz (2 dimensões)

```c
#include <stdio.h>
#include <stdlib.h>
int main(){
    int **p; //2 "*" = 2 níveis = 2 dimensões
    int i, j, N = 2;
    // abaixo defino a quantidade de linhas
    // da matriz
    p = malloc(N*sizeof(int *));

    for (i = 0; i < N; i++){
        // agora para cada linha, defino a
        // quantidade de colunas
        p[i] = malloc(N*sizeof(int));
        for (j = 0; j < N; j++)
            scanf("%d", &p[i][j]);
    }
    return 0;
}
```



Prof. Manoel Pereira Junior
IFMG – Campus Formiga

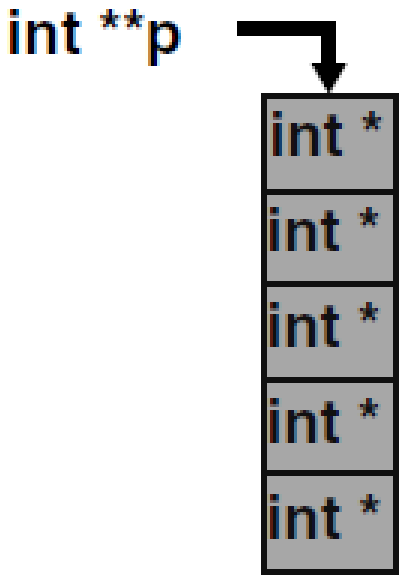
Desalocando uma matriz (2 dimensões)

```c
#include <stdio.h>
#include <stdlib.h>
int main(){
    int **p; //2 "*" = 2 níveis = 2 dimensões
    int i, j, N = 2;
    p = malloc(N*sizeof(int *));
    for (i = 0; i < N; i++){
        p[i] = malloc(N*sizeof(int));
        for (j = 0; j < N; j++)
            scanf("%d",&p[i][j]);
    }
    for (i = 0; i < N; i++){
        free(p[i]);
    }
    free(p);
    return 0;
}
```

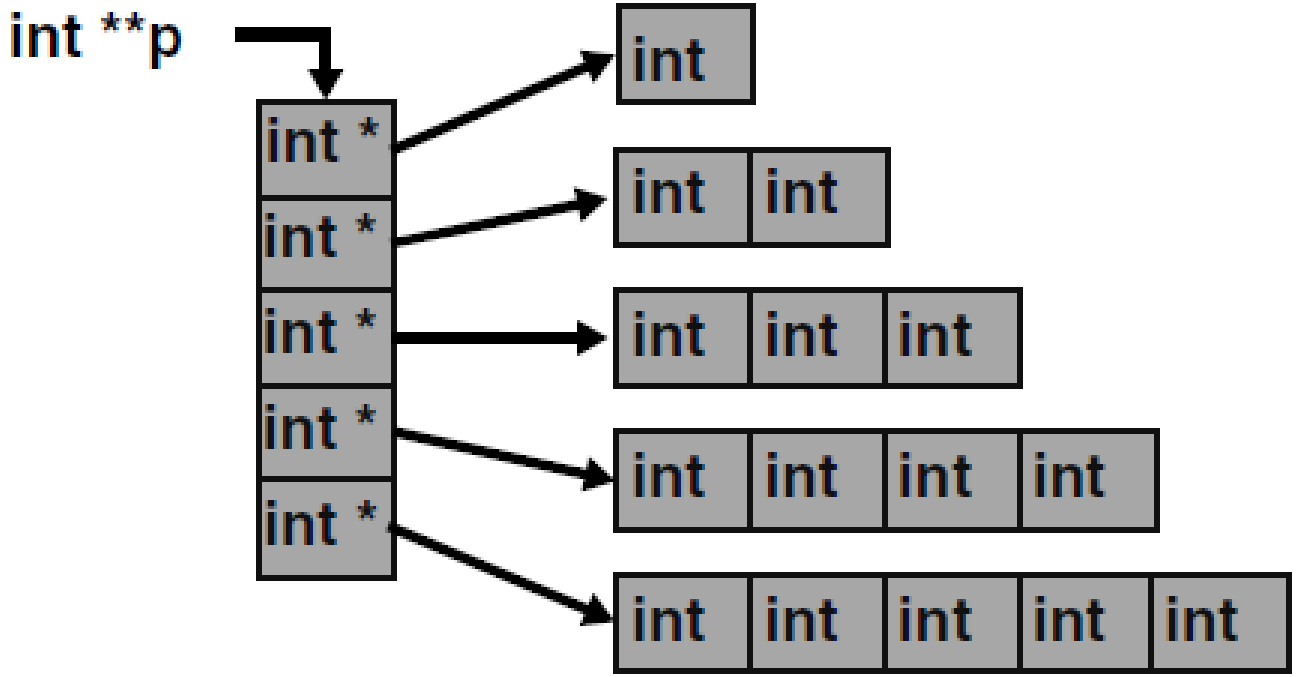
Prof. Manoel Pereira Junior
IFMG – Campus Formiga

Outro exemplo...

1º malloc
Cria as linhas da matriz



2º malloc
Cria as colunas da matriz



Prof. Manoel Pereira Junior
IFMG – Campus Formiga

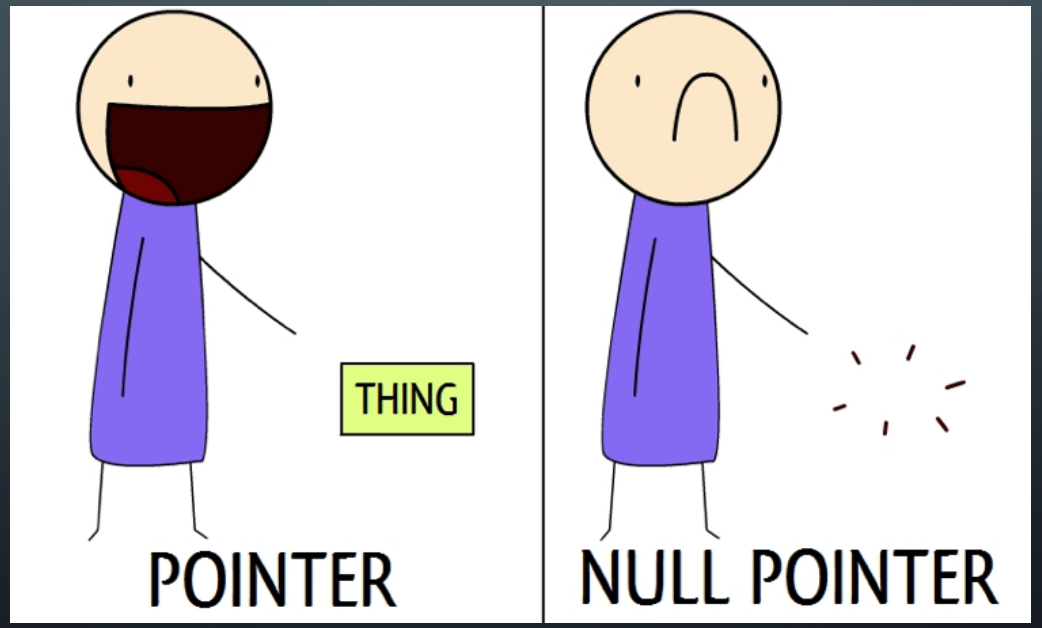
Alocando um vetor de structs dinamicamente...

```c
#define tam 20

int main(){

    ponto *vet_ponto;

    vet_ponto = malloc(tam*sizeof(ponto));

    for (int i = 0; i < tam; i++) {
        vet_ponto[i].x = i*2;
        vet_ponto[i].y = i*3;
    }

    for (int i = 0; i < tam; i++) {
        printf("Vetor na posicao %d\n",i);
        printf("\tValor de X na posição: %d\n",vet_ponto[i].x);
        printf("\tValor de Y na posição: %d\n",vet_ponto[i].y);
    }

    return 0;
}
```

I Pereira Junior
IFMG – Campus Formiga

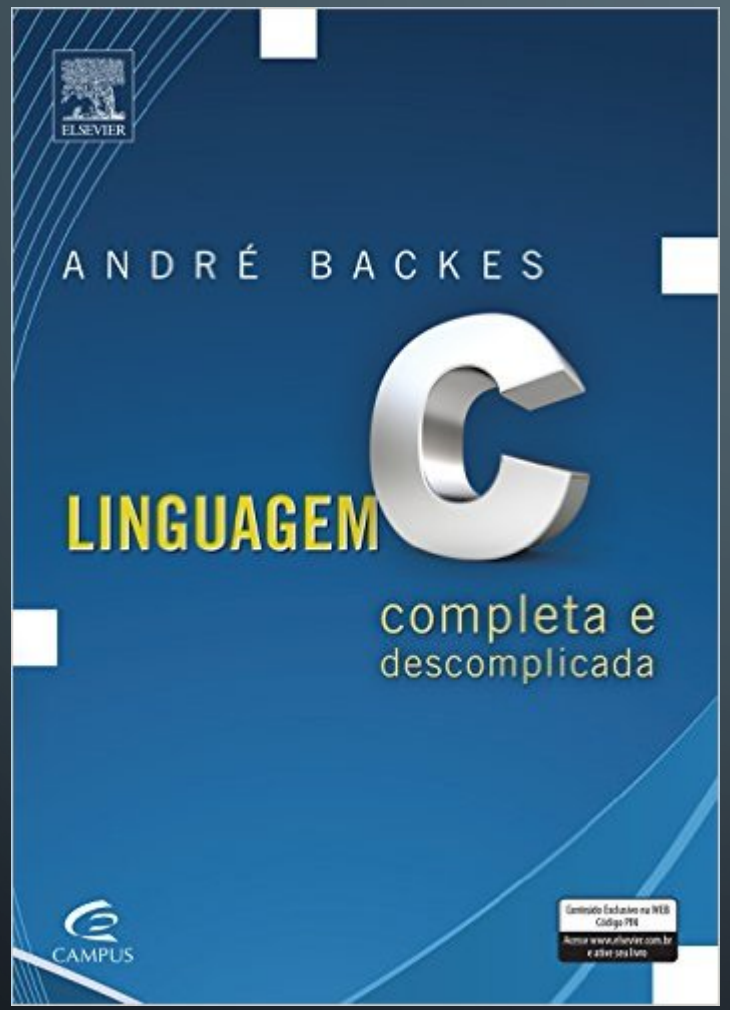
EXERCÍCIOS

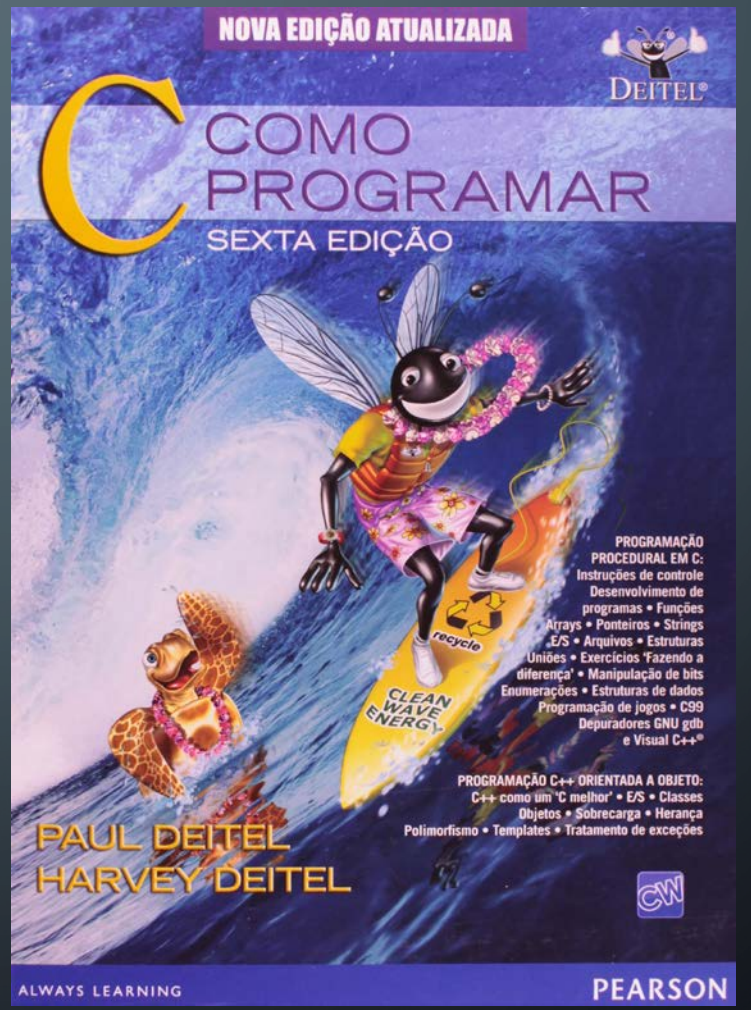
- Linguagem C completa e descomplicada.
- Capítulo 11



Prof. Manoel Pereira Junior
IFMG – Campus Formiga

- Referências:





Prof. Manoel Pereira Junior
IFMG – Campus Formiga