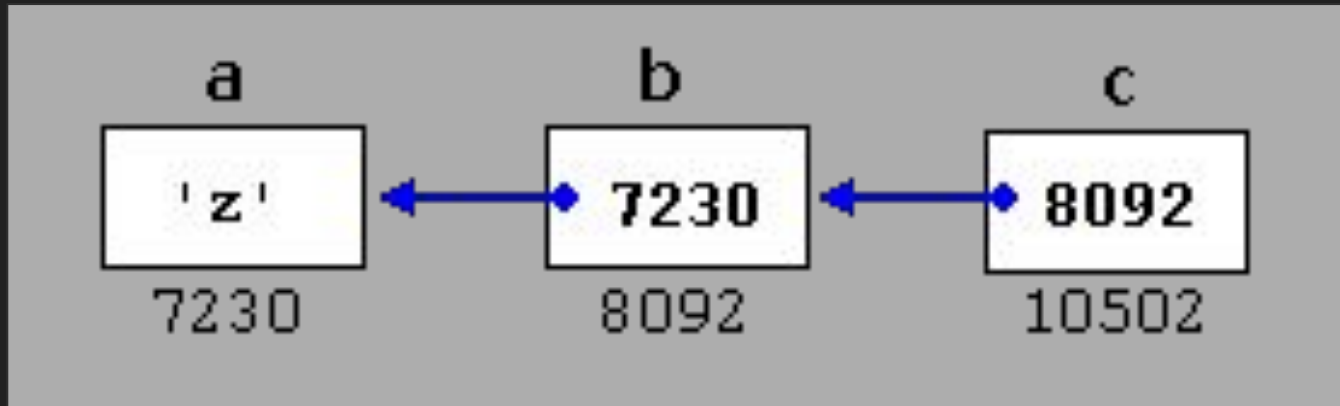


Alocação Dinâmica

Prof.: Leonardo Tórtoro Pereira

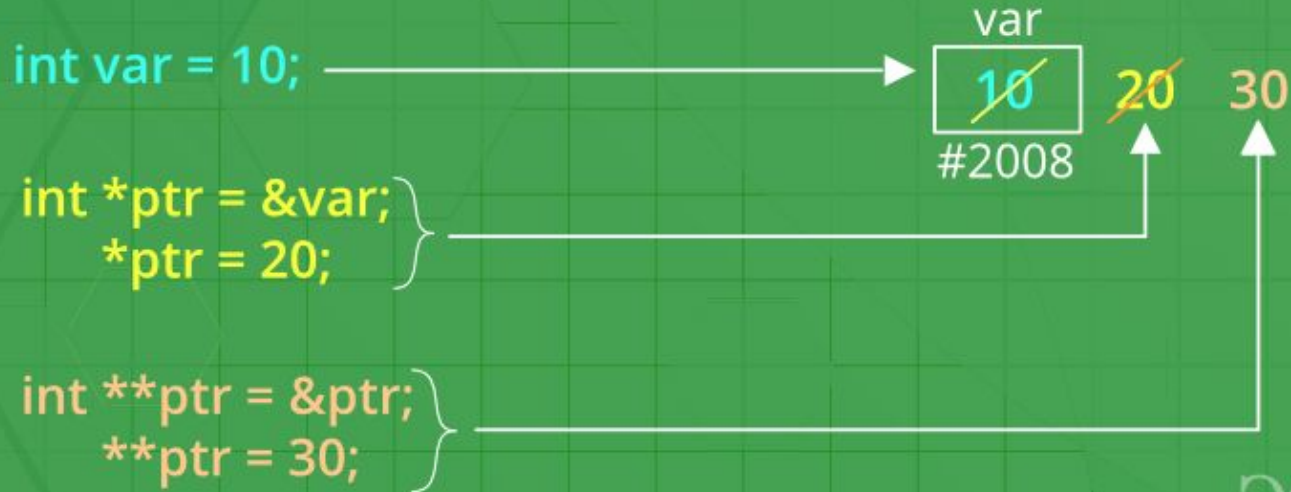
leonardop@usp.br

Ponteiros?



Fonte: <http://www.cplusplus.com/doc/tutorial/pointers/>

How pointer works in C



Alocação Dinâmica

Alocação dinâmica [4]

- Ponteiros também são usados como um receptáculo para regiões de memória alocadas dinamicamente
- Isso é muito útil quando desejamos usar vetores, strings, matrizes, etc. de tamanhos variados, ou dos quais não sabemos o tamanho a tempo de compilação
- Para tal, temos 4 funções da *stdlib*
 - ◆ *malloc()*, *calloc()*, *free()* e *realloc()*
- Hoje vamos falar sobre *malloc*, *calloc* e *free* :)

Alocação dinâmica [4]

- *malloc()* vem de “*memory allocation*”, e serve para alocar um único bloco de memória com o tamanho especificado
- Retorna um ponteiro de tipo *void* que pode receber *cast* para um ponteiro de qualquer forma
- `ptr = (cast-type*) malloc(byte-size)`
- É costume usar o *sizeof()* para indicar o tamanho do tipo a ser alocado, e multiplicá-lo pela quantidade de unidades daquele tipo que deseja-se alocar

Malloc()

```
int* ptr = ( int* ) malloc ( 5* sizeof ( int ) );
```

4 bytes

ptr =



← 20 bytes of memory →

A large 20 bytes memory block is dynamically allocated to ptr



Ponteiros [4]

```
int main() {
    int *ptr, n = 5, i;
    printf("Enter number of elements: %d\n", n);
    ptr = (int*)malloc(n * sizeof(int));
    if (ptr == NULL) {
        printf("Memory not allocated.\n");
        exit(0);
    }
    else {
        printf("Memory successfully allocated using malloc.\n");
        for (i = 0; i < n; ++i)
            ptr[i] = i + 1;
        printf("The elements of the array are: ");
        for (i = 0; i < n; ++i)
            printf("%d, ", ptr[i]);
    }
    free (prt);
    return 0;
}
```

Alocação dinâmica [4]

- Se a alocação falhar (caso não tenha mais memória disponível) ela retorna um ponteiro *NULL*
 - ◆ É importante verificar isso sempre!
- Caso deseje-se inicializar todos os valores com 0, a função *calloc()* faz exatamente isso
 - ◆ A sintaxe é similar à do *malloc()*, exceto que o tamanho de cada elemento é passado separadamente
 - ◆ `ptr = (cast-type*)calloc(n, element-size);`

Alocação dinâmica [4]

- Deve-se “desalocar” TODA memória alocada dinamicamente com o método *free()*.
 - ◆ O programa não libera ela automaticamente!
- Fazer isso para todos os “níveis” de ponteiros alocados
 - ◆ Ver próxima seção

Calloc()

```
int* ptr = ( int* ) calloc ( 5, sizeof ( int ) );
```



Ponteiros [4]

```
int main() {
    int *ptr, n = 5, i;
    printf("Enter number of elements: %d\n", n);
    ptr = (int*)calloc(n * sizeof(int));
    if (ptr == NULL) {
        printf("Memory not allocated.\n");
        exit(0);
    }
    else {
        printf("Memory successfully allocated using malloc.\n");
        printf("The elements of the array are: ");
        for (i = 0; i < n; ++i)
            printf("%d, ", ptr[i]);
    }
    free(ptr);
    return 0;
}
```

Free()

```
int* ptr = ( int* ) calloc ( 5, sizeof ( int ) );
```

4 bytes



operation on ptr

free(ptr)



The memory of ptr is released



Alocação dinâmica [5]

- A função *free()* apenas libera o espaço de memória reservado pelas funções de alocação
 - ◆ O espaço fica disponível para que outras chamadas para tais funções possam usá-lo
 - ◆ Mas o ponteiro ainda aponta para a mesma região de memória
 - Ela só não é mais válida!

Alocação dinâmica [5]

- Se o ponteiro que é passado para *free()* for NULL
 - ◆ A função não faz nada
- Se ele não aponta para um bloco de memória reservado pelas funções de alocação de memória
 - ◆ Causa comportamento indefinido

realloc()

Alocação dinâmica [6]

- O método *realloc(ptr, size)* muda o tamanho do bloco de memória que o ponteiro *ptr* aponta para
- Caso não seja possível estender o bloco atual para o tamanho desejado, a função move o bloco para outro local
 - ◆ Por isso ela retorna um ponteiro!

Alocação dinâmica [6]

- O conteúdo no bloco de memória é preservado até o menor tamanho entre o antigo e o novo, mesmo se ele for movido
 - ◆ Se o novo bloco for maior, os blocos novos tem valor indeterminado
- Se o ponteiro *ptr* for NULL, a função age como o *malloc*

```
// Exemplo em http://www.cplusplus.com/reference/cstdlib/realloc/
int main () {
    int input,n, count = 0, *numbers = NULL, *more_numbers = NULL;
    do {
        printf ("Enter an integer value (0 to end): ");
        scanf ("%d", &input);
        count++;
        more_numbers = (int*) realloc (numbers, count * sizeof(int));
        if (more_numbers!=NULL) {
            numbers=more_numbers;
            numbers[count-1]=input;
        }
        else {
            free (numbers);
            puts ("Error (re)allocating memory");
            exit (1);
        }
    } while (input!=0);
    printf ("Numbers entered: ");
    for (n=0;n<count;n++) printf ("%d ",numbers[n]);
    free (numbers);
    return 0;
}
```

Realloc()

```
int* ptr = ( int* ) malloc ( 5* sizeof ( int ));
```

4 bytes

ptr =  → A large 20 bytes memory block is dynamically allocated to ptr

← 20 bytes of memory →

```
ptr = realloc ( ptr, 10* sizeof( int ));
```

ptr =  → The size of ptr is changed from 20 bytes to 40 bytes dynamically

← 40 bytes of memory →



Alocação dinâmica [4]

- Quando usamos *realloc* em um ponteiro passado para uma função corremos o risco de perder a referência original dele fora do escopo da função!
- Não devemos dar *realloc* em ponteiros de função se não retornarmos este (possível) novo ponteiro
- Uma alternativa é passar um ponteiro para o ponteiro que será realocado!

Exemplo Realloc em Funções

```
//Retornando o ponteiro realocado
char* concatStrings(char* string1, char* string2) {
    //String1 + String2 + \n + espaço
    int newSize = strlen(string1) + strlen(string2) + 2;
    int index1 = strlen(string1);
    string1 = (char*) realloc(string1, sizeof(char)*newSize);
    string1[index1++] = ' ';
    int index2 = 0;
    while(string2[index2] != '\0') {
        string1[index1++] = string2[index2++];
    }
    string1[index1] = '\0';
    return string1;
}
```

Exemplo Realloc em Funções

```
int main(){
    char* stringPtr = NULL;
    char string1[] = "fique";
    char string2[] = "em";
    char string3[] = "casa";
    stringPtr = (char*) malloc(sizeof(char)*strlen(string1));
    strcpy(stringPtr, string1);
    printf("\n%s\n", stringPtr);
    stringPtr = concatStrings(&stringPtr, string2);
    printf("\n%s\n", stringPtr);
    stringPtr = concatStrings(&stringPtr, string3);
    printf("\n%s\n", stringPtr);
    return 0;
}
```


Alternativa também correta:

```
//Usando ponteiro para o ponteiro a ser realocado
void concatStrings2(char** string1, char* string2){
    //String1 + String2 + \n + espaço
    int newSize = strlen(*string1) + strlen(string2) + 2;
    int index1 = strlen(*string1);
    *string1 = (char*) realloc(*string1, sizeof(char)*newSize);
    (*string1)[index1++] = ' ';
    int index2 = 0;
    while(string2[index2] != '\0') {
        (*string1)[index1++] = string2[index2++];
    }
    (*string1)[index1] = '\0';
}
```

Alternativa errada:

```
//Não retornando o ponteiro realocado - ERRADO!!!  
void concatStringsWrong(char* string1, char* string2){  
    //String1 + String2 + \n + espaço  
    int newSize = strlen(string1) + strlen(string2) + 2;  
    int index1 = strlen(string1);  
    string1 = (char*) realloc(string1, sizeof(char)*newSize*100);  
    string1[index1++] = ' ';  
    int index2 = 0;  
    while(string2[index2] != '\0')    {  
        string1[index1++] = string2[index2++];  
    }  
    string1[index1] = '\0';  
}
```

Referências

1. <http://www.cplusplus.com/doc/tutorial/pointers/>
2. <https://www.geeksforgeeks.org/pointer-vs-array-in-c/>
3. <https://www.geeksforgeeks.org/pointers-in-c-and-c-set-1-introduction-arithmetic-and-array/>
4. <https://www.geeksforgeeks.org/dynamic-memory-allocation-in-c-using-malloc-calloc-free-and-realloc/>
5. <http://www.cplusplus.com/reference/cstdlib/free/>
6. <http://www.cplusplus.com/reference/cstdlib/realloc/>