Ponteiros e Alocação Dinâmica

Introdução

Prof. Fernando Figueira
(adaptado do material do Prof. Rafael Beserra Gomes)

UFRN

Material compilado em 13 de outubro de 2025.

Endereço base

0xbffff22c	0	0	0	0	0	1	0	1	**short int** a = 5
0xbffff22d	0	1	0	0	0	0	1	0	**char** b = 'B'
0xbffff22e	0	1	0	0	0	0	1	1	**char** x = 'C'
0xbffff22f	1	1	0	1	1	1	0	1	**float** y = 3.2
0xbffff230	1	1	0	0	1	1	0	0	
0xbffff231	0	1	0	0	1	1	0	0	
0xbffff232	0	1	0	0	0	0	0	0	
0xbffff233	0	1	0	0	0	0	0	1	**char** t1 = 'A'
0xbffff234	0	1	0	0	0	0	1	0	**char** t2 = 'B'
0xbffff235	0	1	0	0	0	0	1	1	**char** t3 = 'C'
0xbffff236	0	1	0	0	0	0	1	1	**char** t4 = 'A'
0xbffff237	1	1	1	1	1	0	1	1	**short int** u = -5
0xbffff238	0	0	0	0	0	0	0	0	**int** v = 1
0xbffff239	0	0	0	0	0	0	0	0	
0xbffff23a	0	0	0	0	0	0	0	0	
0xbffff23b	0	0	0	0	0	0	0	1	

endereço base da variável y

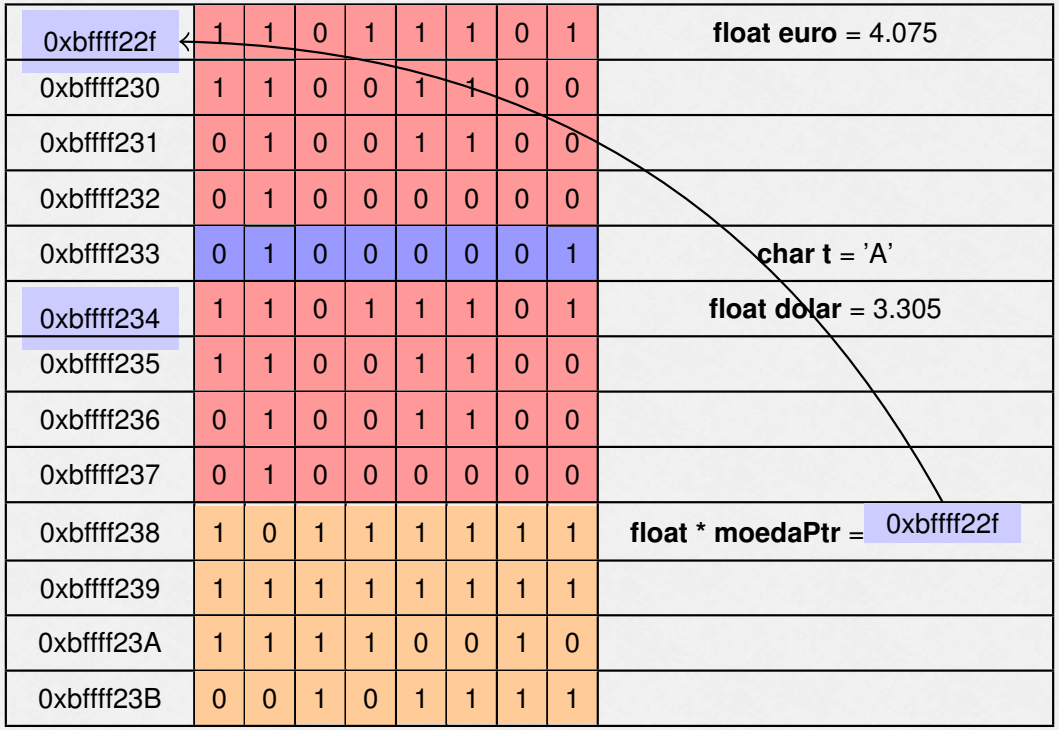
Ponteiros

Definição

Uma variável que contém como valor um endereço de memória.

A quantidade de bits necessária para representar um endereço de memória depende da arquitetura do computador e do sistema operacional.

Exemplo de ponteiro


0xbffff22f 1 1 0 1 1 1 0 1 float euro = 4.075
0xbffff230 1 1 0 0 1 1 0 0
0xbffff231 0 1 0 0 1 1 0 0
0xbffff232 0 1 0 0 0 0 0 0
0xbffff233 0 1 0 0 0 0 0 1 char t = 'A'
0xbffff234 1 1 0 1 1 1 0 1 float dolar = 3.305
0xbffff235 1 1 0 0 1 1 0 0
0xbffff236 0 1 0 0 1 1 0 0
0xbffff237 0 1 0 0 0 0 0 0
0xbffff238 1 0 1 1 1 1 1 1 float * moedaPtr = 0xbffff22f
0xbffff239 1 1 1 1 1 1 1 1
0xbffff23A 1 1 1 1 0 0 1 0
0xbffff23B 0 0 1 0 1 1 1 1

Ponteiro para char:

0xbffff22d	0	1	0	0	0	0	1	0	**char** b = 'B'
0xbffff22e	0	1	0	0	0	0	1	1	**char** x = 'C'
0xbffff22f	1	1	0	1	1	1	0	1	**float** y = 3.2
0xbffff230	1	1	0	0	1	1	0	0	
0xbffff231	0	1	0	0	1	1	0	0	
0xbffff232	0	1	0	0	0	0	0	0	
0xbffff233	0	1	0	0	0	0	0	1	**char** t1 = 'A'
0xbffff234	0	1	0	0	0	0	1	0	**char** t2 = 'B'
0xbffff235	0	1	0	0	0	0	1	1	**char** t3 = 'C'
0xbffff236	0	1	0	0	0	0	1	1	**char** t4 = 'A'
0xbffff237	1	1	1	1	1	0	1	1	**short int** u = -5
0xbffff238	1	0	1	1	1	1	1	1	**char** * p = 0xbffff234
0xbffff239	1	1	1	1	1	1	1	1	
0xbffff23a	1	1	1	1	0	0	1	0	
0xbffff23b	0	0	1	1	0	1	0	0	

Implementação

Implementação

Implementação

Declarando ponteiros:

```
float euro = 4.075, dolar = 3.305;
float *moedaPtr;
int a, *b, c;
```

- **a** e **c**: tipo int
- **b**: **ponteiro para inteiro**

Implementação

Atribuindo valores:

```
float euro = 4.075, dolar = 3.305;
float *moedaPtr;
int a, *b, c;

b = 0x7fffffffddd4;
moedaPtr = &euro;
```

- é possível atribuir manualmente um endereço (não usual)
- operador &: obtém o endereço da variável

Implementação

Atribuindo valores:

```
float euro = 4.075, dolar = 3.305;
float *moedaPtr;

moedaPtr = NULL;
```

NULL: o ponteiro não está armazenando um endereço válido

- bastante útil em estruturas de dados:

Implementação

Escrevendo na tela o valor de um ponteiro:

```
int *a, b;

a = &b;
printf("Endereco: %p\n", a);
```

Use %p no printf. Observe que é atribuído um endereço diferente a cada execução do programa.

Implementação

✎ Exercício em sala

1. Declare dois inteiros: a e b (valores iniciais: 2 e 3)
2. Declare três ponteiros para inteiro: p1, p2, p3
3. Atribua NULL para p1
4. Atribua o endereço de a para p2
5. Atribua o endereço de b para p3
6. Escreva na tela duas vezes o endereço de a: usando o operador & e usando o valor do ponteiro
7. Escreva na tela duas vezes o endereço de b: usando o operador & e usando o valor do ponteiro

Implementação

✎ Exercício em sala

Atribua um valor ao ponteiro p de forma que o printf escreva somente **"uma string com varias palavras"**.

```
char string[] = "Esta eh uma string com varias palavras"

char *p = ;

printf("%s\n", p);
```

Derreferenciando um ponteiro:

```
float euro = 4.075, dolar = 3.305;
float *moedaPtr;

moedaPtr = &euro;
printf("Cotacao do euro: %f\n", *moedaPtr);
moedaPtr = &dolar;
printf("Cotacao do dolar: %f\n", *moedaPtr);
```

Derreferenciar um ponteiro: obter o valor no endereço de memória armazenado no ponteiro
Uso: operador * antes do ponteiro.

Aritmética de ponteiros

Aritmética de ponteiros:

Suponha um vetor declarado como: **int x[4];**

0xbffff228	0 0 0 0 0 0 0 0	**x[0]** = 15
0xbffff229	0 0 0 0 0 0 0 0	
0xbffff22a	0 0 0 0 0 0 0 0	
0xbffff22b	0 0 0 0 1 1 1 1	
0xbffff22c	0 0 0 0 0 0 0 0	**x[1]** = 10
0xbffff22d	0 0 0 0 0 0 0 0	
0xbffff22e	0 0 0 0 0 0 0 0	
0xbffff22f	0 0 0 0 1 0 1 0	
0xbffff230	0 0 0 0 0 0 0 0	**x[2]** = 3
0xbffff231	0 0 0 0 0 0 0 0	
0xbffff232	0 0 0 0 0 0 0 0	
0xbffff233	0 0 0 0 0 0 1 1	
0xbffff234	0 0 0 0 0 0 0 0	**x[3]** = 1
0xbffff235	0 0 0 0 0 0 0 0	
0xbffff236	0 0 0 0 0 0 0 0	
0xbffff237	0 0 0 0 0 0 0 1	
0xbffff238	1 0 1 1 1 1 1 1	**int * p** = x (0xbffff228)
0xbffff239	1 1 1 1 1 1 1 1	
0xbffff23a	1 1 1 1 0 0 1 0	
0xbffff23b	0 0 1 1 0 1 0 0	

- p+1 corresponde a 0xbffff22c
- p+2 corresponde a 0xbffff230
- *(p+1) é 10 (o mesmo que p[1])
- *(p+2) é 3 (o mesmo que p[2])

Calculando o tamanho de uma string:

```
#include <stdio.h>

int main() {

        char string[] = "Esta eh uma string de teste";
        int cont = 0;

        char *p = string;
        while(*p != '\0') {
                cont++;
                p++;
        }
        printf("Tamanho da string: %d\n", cont);

        return 0;
}
```

✎ Exercício em sala

Usando ponteiros, escreva um programa que leia uma string e escreva na tela a quantidade de palavras na string. Uma palavra aqui é definida como qualquer combinação de caracteres diferente de espaço.

Alocação dinâmica

Definição

Definição

Alocação dinâmica significa reservar espaço na memória durante a execução do programa

Isso permite, por exemplo, utilizar um vetor de tamanho variável.

Implementação

Alocando um vetor dinamicamente:

```
int *u, *v;
u = malloc(4*sizeof(int));
v = calloc(4, sizeof(int));
```

malloc:

- parâmetro: número de bytes
- retorno: endereço base do espaço reservado (NULL em caso de falha)

calloc:

- parâmetros: número de elementos, tamanho do tipo
- retorno: endereço base do espaço reservado (NULL em caso de falha)
- espaço é zerado

Implementação

Acessando o espaço alocado dinamicamente:

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

int main() {
        int *v;
        v = malloc(4*sizeof(int));
        v[0] = 4;
        v[1] = 7;
        v[2] = 3;
        v[3] = 1;
        v[4] = 9;
        return 0;
}
```

O que ocorre ao atribuir 9 a v[4]?

Implementação

✎ Exercício em sala

Escreva um programa para alocar dinamicamente espaço para 5 inteiros na memória, preencha-o com os inteiros 1, 3, 5, 7 e 9. Depois escreva os mesmos valores na tela usando um **for**.

Implementação

Liberando a memória:

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

int main() {
        int *v;
        int n, i;

        scanf("%d", &n);
        v = malloc(n*sizeof(int));

        for(i = 0; i < n; i++) {
                scanf("%d", &v[i]);
        }

        free(v);
        return 0;
}
```

Implementação

Liberando a memória:

- O sistema operacional deve liberar qualquer memória alocada (dinamicamente ou de forma estática) quando o programa encerrar
- Deixar de liberar a memória alocada durante a execução de um programa pode levar a *memory leak*

Implementação

✎ Exercício em sala

Escreva um programa que leia um inteiro **n**, **n** inteiros e, em seguida, um inteiro **x**. O programa deve escrever na tela quantos dos **n** inteiros são iguais a **x**. Não deixe de usar **sizeof** e **free**.

Uso de ponteiros em funções

Uso de ponteiros em funções

Uso de ponteiros em funções

Quando uma função é chamada, os argumentos são **copiados** para os parâmetros (não se referem ao mesmo dado na memória!)

```
void f(int x) {
        x--;
}

int main() {
        int k;
        k = 3;

        printf("k = %d\n", k);
        f(k);
        printf("k = %d\n", k);
        return 0;
}
```

Uso de ponteiros em funções

Usando ponteiro para alterar a variável da main:

```
void f(int *x) {
        *x = *x - 1;
}

int main() {
        int k;
        k = 3;

        printf("k = %d\n", k);
        f(&k);
        printf("k = %d\n", k);

        return 0;
}
```