



GOVERNO DO ESTADO DO RIO DE JANEIRO
UERJ – UNIVERSIDADE DO ESTADO DO RIO DE JANEIRO - ZO
CTC – CENTRO DE TECNOLOGIAS E CIÊNCIAS
FCEE – FACULDADE DE CIÊNCIAS EXATAS E ENGENHARIAS
DEPCOMP – DEPARTAMENTO DE COMPUTAÇÃO
CURSOS DE COMPUTAÇÃO

Estrutura de Dados I
Parte 3 – Ponteiros.
Prof. Raul Queirós

Apontadores: Ponteiros

Estrutura de Dados I

Apontadores (ponteiros).








A linguagem C permite que armazenemos em uma posição de memória o endereço de uma variável (referência a uma célula de memória). Deste modo, na verdade criamos uma variável que é um ponteiro para uma determinada posição de memória (acesso indireto).

Com este recurso, a linguagem C torna-se mais flexível, permitindo a execução de instruções que só eram possíveis em linguagem de montagem (ex: assembler).

Cada posição de memória tem um endereço único capaz de referenciá-la na memória. Cada variável criada está relacionada a um endereço de memória. Assim, não existem variáveis com o mesmo endereço.

Estrutura de Dados I

Se observarmos a memória, ao criarmos nossas variáveis teríamos:

Tipo	Células de Memória	Tamanho
char		1 byte
byte		1 byte
short		2 bytes
int		4 bytes
long		4/8 bytes
float		4 bytes
double		8 bytes

Estrutura de Dados I

```
#include "stdio.h"
```

```
int main(){  
    printf("\nTamanho      char= %d.", sizeof(char));  
    //printf("Tamanho      byte= %d.", sizeof(byte));  
    printf("\nTamanho      short= %d.", sizeof(short));  
    printf("\nTamanho      int= %d.", sizeof(int));  
    printf("\nTamanho      long= %d.", sizeof(long));  
    printf("\nTamanho      float= %d.", sizeof(float));  
    printf("\nTamanho      double= %d.", sizeof(double));  
    printf("\nTamanho      longdouble= %d.", sizeof(long double));  
  
    return 0;  
}
```

Estrutura de Dados I

```
E:\OneDrive\2020-2021\2021\Carlos_2021-2\Trab_UV\EstruturasDados\exemplos\Parte3Exemplo1.exe

Tamanho      char= 1.
Tamanho      short= 2.
Tamanho      int= 4.
Tamanho      long= 4.
Tamanho      float= 4.
Tamanho      double= 8.
Tamanho longdouble= 12.
-----
Process exited after 0.1906 seconds with return value 0
Pressione qualquer tecla para continuar. . .
```

Estrutura de Dados I

Um programa executável é dividido em três regiões distintas:

1. Área de Código – onde ficam armazenadas as instruções do programa;
2. Área de Dados – onde ficam armazenadas as variáveis globais necessárias à execução do programa;
3. Área de Registros de Ativação – onde serão armazenadas as variáveis locais das funções. Estas variáveis só terão um espaço de memória, na ativação da função. Deste modo, quando a função termina sua execução este espaço é liberado e quando a função é chamada novamente, estas variáveis serão alocadas na memória novamente. Assim, podemos entender porque ao retornarmos de uma função, as suas variáveis locais não estão mais disponíveis para consulta.

Estrutura de Dados I

Em um determinado compilador:

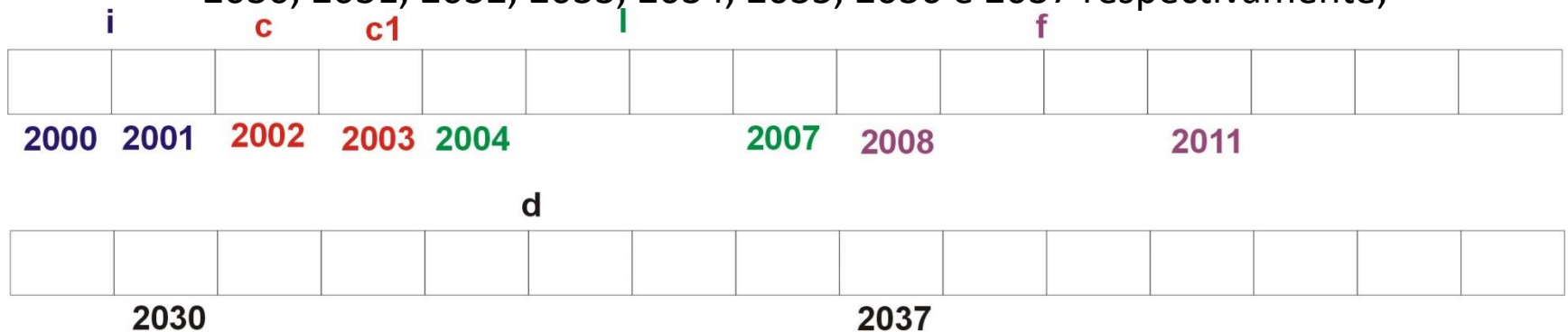
A variável *i* encontra-se na memória (**com valores convertidos para decimais**), no endereço 2000, mas, como ela possui **2 bytes** por ser um inteiro, ocupa os endereços 2000 e 2001 respectivamente;

A variável *c* encontra-se na memória, no endereço 2002, pois, ela possui **1 byte**, assim como a variável *c1* em 2003;

A variável *l* encontra-se na memória, no endereço 2004, mas, como ela possui **4 bytes** por ser um inteiro longo, ocupa os endereços 2004, 2005, 2006 e 2007 respectivamente;

A variável *f* encontra-se na memória, no endereço 2008, mas, como ela possui **4 bytes** por ser um ponto flutuante, ocupa os endereços 2008, 2009, 2010 e 2011 respectivamente;

A variável *d* encontra-se na memória, no endereço 2030, mas, como ela possui 8 bytes por ser um ponto flutuante de máxima precisão, ocupa os endereços 2030, 2031, 2032, 2033, 2034, 2035, 2036 e 2037 respectivamente;



Obs.: Quando o programa é compilado pelo programador, os nomes das variáveis são substituídos pelo endereço da variável é dado pelo primeiro endereço da primeira posição da variável na memória (endereço base da variável).

Estrutura de Dados I

O uso de apontadores, dá a linguagem C maior flexibilidade, possibilitando-o executar instruções até então restritas aos montadores(assembler). Como vantagem extra, os códigos gerados pelo compilador são mais eficientes e compactos.

Estrutura de Dados I

Declaração de apontadores.

A declaração é semelhante a declaração de variáveis, acrescentando apenas um * (asterisco) na frente do nome da variável.

<tipo> * <nome>;

Exemplos:

```
int *i;           // cria um ponteiro para uma variável inteira
char* c;          // cria um ponteiro para uma variável caractere
long * l;         // cria um ponteiro para uma variável inteira longa
float *f;         // cria um ponteiro para uma variável em ponto flutuante
double *d;        // cria um ponteiro para uma variável ponto flutuante em
                  // máxima precisão
```

Obs.: Caso seja usado um ponteiro de um determinado tipo (ex. int) para uma variável de outro tipo (ex. real), o programa não mostrará erro de compilação, mas, o resultado será totalmente inesperado.

Estrutura de Dados I

Atribuição de Apontadores

Para fazer referência diretamente aos endereços das variáveis, utilizamos o operador & na frente da variável.

Exemplo: &c;

Para indicarmos a posição da variável (endereço) onde esta está alocada, usamos:

```
{ ...  
    int v, *pv;  
    pv = &v;  
    .....  
}
```

Após a execução do código acima, o conteúdo de pv, será o endereço da variável v.

```
#include "stdio.h"
```

```
int main(){
```

```
    int idade=15, *pidade;
```

```
    pidade = &idade;
```

```
    printf("\nEnd.  idade %p, valor de idade %d.", &idade, idade);
```

```
    printf("\nEnd.  pidade %p, valor de pidade %p.", &pidade, pidade);
```

```
    printf("\nEnd. valor de idade %d, via ponteiro pidade.", *pidade);
```

```
    printf("\nPróximo End. pidade: %p", (pidade)++);
```

```
return 0;
```

```
}
```

```
Selecionar E:\OneDrive\2020-2021\2021\Carlos_2021-2\Trab_UV\EstruturasDados\exemplos\Parte3Exemplo2.exe

End. idade 0063fecc, valor de idade 15.
End. pidade 0063fec8, valor de pidade 0063fecc.
End. valor de idade 15, via ponteiro pidade.
Próximo End. pidade: 0063fecc
-----
Process exited after 0.1547 seconds with return value 0
Pressione qualquer tecla para continuar. . .
```

Exemplo de programa em C:

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

```
main( ) {
```

```
    char ch, *pch, v; // Declaramos as variáveis para caracteres, ch, v e um  
                      // ponteiro para caracteres pch
```

```
    ch = 'A';         // Atribuímos 'A' à variável ch
```

```
    pch = &ch;        // Atribuímos ao apontador pch, o endereço da variável ch
```

```
    printf("%c", *pch); // Apresentamos o conteúdo da variável ch, através do  
                        // endereço do apontador pch.
```

```
    v = *pch;         // Atribuição do conteúdo de ch em v, via o endereço  
                      // armazenado em pch.
```

```
}
```

Estrutura de Dados I

Aritméticas com Apontadores:

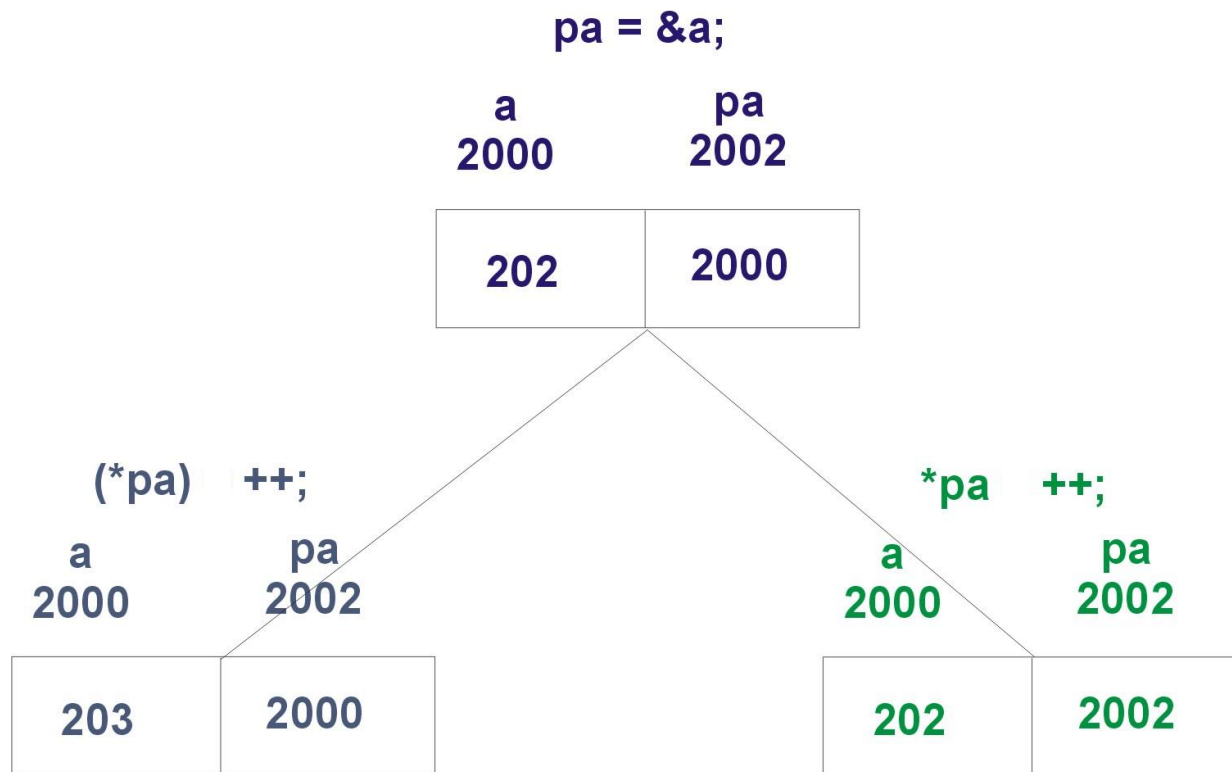
```
int x,y, *py, *px; // Declaramos a variável para inteiros y e dois ponteiros py e px;
py = &y; px=&x;    // Atribuímos ao apontador py, o endereço da variável y;
*px = 3;           // Atribuímos o valor 3 a variável x;
*px +=1; // Incrementamos a variável x de 1 unidade, armazenando o valor: 3 + 1 =4.
```

Apontando diretamente um endereço de memória:

```
char *px, *py;
px = py = 0xf228; //Atribuição de um endereço em hexadecimal.
                // Ambos os apontadores apontarão para f228 hexadecimal
```


Estrutura de Dados I

Analizando o exemplo abaixo,
`int a = 202, *pa;`



Observamos a diferença de incrementarmos o conteúdo do endereço apontado por pa e incrementarmos o apontador pa.

Estrutura de Dados I

Exemplo de exibição de conteúdo utilizando-se apontadores:

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

```
main()
```

```
{
```

```
    int a=5, *pa1, *pa2;
```

```
    pa1 = pa2 = &a;
```

```
    printf("Endereço de pa1= %x, e conteúdo apontado por  
pa1=%d", pa1, *pa1);
```

```
}
```

Obs.: É possível a comparação entre ponteiros, desde que ambos sejam do mesmo tipo.

$x \leq y$ // $>$, $<$, \geq , \leq , $==$, $!=$

Estrutura de Dados I

Erros comuns ao se utilizar apontadores:

```
int a, *pa;
```

```
&a = &a + 1; // tentativa de alterar o endereço da variável a.
```

```
pa = &(a+1); // tentativa de tentar saber o próximo endereço após a  
            // variável a.
```

```
pa = &(3); // tentativa de buscar o endereço de uma constante de  
           // valor = 3.
```

Estrutura de Dados I

Expressões com Apontadores:

Apontadores podem ser usados como variáveis em expressões aritméticas.

Exemplos:

```
int *pz, y, z=20;
```

```
pz = &z;
```

```
y = *pz + 1;
```

```
// O operador * tem precedência sobre a soma +, assim, o conteúdo do  
// endereço de pz será somado a 1 e o resultado armazenado em y.
```

Ao final: z será igual a 20 e y será igual a 21.

Estrutura de Dados I

Analisando o código abaixo:

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

```
main( ) {
```

```
    int a = 10, b, *pa;
```

```
    pa = &a;
```

```
    b = *pa + 1;    // Nesta instrução, b recebe o conteúdo apontado por pa + 1.
```

```
    printf("Valor de b=%d", b);
```

```
    a = *(pa + 1); // Nesta instrução, a recebe o conteúdo do próximo endereço, após o
```

```
}                // endereço da variável a, no caso o conteúdo da variável b.
```

```
b = *pa + 1;
```

```
b = 10 + 1;
```

```
b = 11;
```

b = *pa + 1;

a 2000	b 2002	pa 2004
10	11	2000

```
a = *(pa + 1);
```

```
a = *(2000 + 1);
```

```
a = *(2002); // próximo endereço inteiro
```

```
a = 11;
```

b = *pa + 1;

a 2000	b 2002	pa 2004
10	11	2000

Estrutura de Dados I

Apontadores Para Vetores:

Em linguagem C, os elementos de um vetor podem ser encarados como variáveis e desta forma qualquer operação sobre vetores pode ser feita exatamente como é feita sobre uma variável comum.

```
int vet[10], *pvet;    // declaramos um vetor vet com 10 elementos
                        // inteiros e um apontador pvet.

pvet = &vet[0];        // pvet aponta para o primeiro elemento do
                        // vetor.

pvet++;                // pvet aponta para o próximo elemento do vetor
                        // vet[1].

*pvet=5;               // o segundo elemento do vetor recebe o valor 5.

printf("%d", vet[1]);  // é apresentado o valor do segundo elemento
                        // de vet, vet[1].
```

Estrutura de Dados I

Uma vantagem para a programação em C, é que ao adicionarmos ou subtrairmos um valor ao endereço do apontador, esta constante é multiplicada primeiro pelo tamanho em bytes do tipo do dado. Assim:

```
int vet[10] = { 2,4,6,8,10,12,14,16,18,20}; // declara-se e inicializa-se o vetor.
```

```
int a, *pvet;    // declara inteiro a e ponteiro para inteiro pvet.
```

```
pvet = &vet[0]; // pvet aponta para o primeiro elemento de vet.
```

```
a = *(pvet + 3); // a recebe o valor do deslocamento de 3 endereços para a  
                // direita, o 4º elemento de vet, vet[3] que vale 8. 3 foi  
                // multiplicado por 2, deslocando de 6 bytes no vetor.
```

Obs.: É possível em função da similaridade de apontadores e vetores o uso da expressão:

```
pvet = vet;
```

Ao invés da segunda instrução `pvet = &vet[0];`

Estrutura de Dados I

Exemplo de uso de apontadores:

```
#include <stdio.h>

main( ) {
    int a, *aPtr;        // declaração de a e do ponteiro aPtr, ambos inteiros.
    a = 7;                // atribuição de 7 a a.
    aPtr = &a;            // aPtr recebe o endereço de a.
    printf("O endereço de a e %p\n O valor de aPtr e %p\n\n", &a, aPtr);
    printf("O valor de a e %d\n O valor de *aPtr e %d\n\n", a, *aPtr);
    printf("Sabendo que * e & complementam-se mutuamente.\n &*aPtr = %p\n *&aPtr = %p\n", &*aPtr, *&aPtr);
}
```

Uma saída de exemplo será:

O endereço de a e FFF4

O valor de aPtr e FFF4

O valor de a e 7

O valor de *aPtr e 7

Sabendo que * e & complementam-se mutuamente.

&*aPtr = FFF4

*&aPtr = FFF4

Exercícios (Parte 3):

1. Mostre na tabela abaixo todos os passos (teste de mesa) e identifique qual será a saída do programa em C, para os valores lidos ($x = 3$ e $y = 4$).

```
void func(int *px, int *py)
{
    px = py;
    *py = (*py) * (*px);
    *px = *px + 2;
}

void main(void)
{
    int x, y;
    scanf("%d",&x); /*3*/
    scanf("%d",&y); /*4/
    func(&x,&y);
    printf("x = %d, y = %d", x, y);
}
```

x	y	px	py

Exercícios (Parte 3):

2. Escreva os valores das variáveis para cada instrução do programa abaixo. Qual a saída do programa?

```
int main(int argc, char *argv[])
{
    int a,b,*p1, *p2;
    a = 4;
    b = 3;
    p1 = &a;
    p2 = p1;
    *p2 = *p1 + 3;
    b = b * (*p1);
    (*p2)++;
    p1 = &b;
    printf("%d %d\n", *p1, *p2);
    printf("%d %d\n", a, b);
}
```

a	b	p1	p2
4	3	&a	

Estrutura de Dados I

3. Crie um programa contendo o seguinte trecho de código:

```
int a = 25;
```

```
int *pa = &a;
```

```
int b = *pa + a;
```

```
printf("%d %d %d %d %d %d\n", a, pa, &a, *pa, b, &b);
```

a) Qual o resultado da execução do trecho do programa?

b) Qual o significado de cada um dos valores escritos na tela?

Estrutura de Dados I

4. Crie um programa para calcular a área e o perímetro de um hexágono. O seu programa deve implementar uma função chamada `calcula_hexagono` que calcule a área e o perímetro de um hexágono regular de lado `L`. A função deve obedecer o seguinte protótipo:

`void calcula_hexagono(float l, float *area, float *perimetro);`

- Lembrando que a área e o perímetro de um hexágono regular são dados por:

$$A = \frac{3l^2\sqrt{3}}{2} \quad P = 6l$$

- Para os cálculos, obrigatoriamente você deve utilizar as funções `sqrt` e `pow` da biblioteca `math.h`.
- Em seguida crie a função principal do programa e utilize a função `calcula_hexagono` para calcular a área e o perímetro de um hexágono de lado `l` informado pelo usuário.

Estrutura de Dados I

5. Escreva uma função que determine a média e a situação de um aluno em uma disciplina. A função recebe como parâmetros as três notas de um aluno (p_1 , p_2 , e p_3), seu número de faltas (faltas), o número total de aulas da disciplina (aulas) e o ponteiro para uma variável (media), conforme o seguinte protótipo:

```
char situacao(float p1, float p2, float p3, int faltas,  
int aulas, float *media);
```

Na variável indicada pelo ponteiro media, a função deve armazenar a média do aluno, calculada como a média aritmética das três provas. Além disso, a função deve retornar um caractere indicando a situação do aluno no curso, definido de acordo com o critério a seguir:

Estrutura de Dados I

Número de Faltas	Média	Situação	Retorno
Menor ou igual a 25% do total de aulas	Maior ou igual 6,0	Aprovado	A
	Menor que 6,0	Reprovado	R
Maior que 25% do total de aulas	Qualquer	Reprovado por faltas	F

Em seguida, escreva a função principal de um programa que utiliza a função anterior para determinar a situação de um aluno. O programa deve:

- Ler do teclado três números reais e dois números inteiros, representando as notas da p1, p2 e p3, o número de faltas e o número de aulas, respectivamente;
- Chamar a função desenvolvida na primeira questão para determinar a média e a situação do aluno na disciplina;
- Exibir a média (com apenas uma casa decimal) e a situação do aluno, isto é, “APROVADO”, “REPROVADO” ou “REPROVADO POR FALTAS”, dependendo do caractere retornado pela função, conforme a tabela acima.

Estrutura de Dados I

Obrigado pela atenção!
Fim da parte 3.