MAC121 - Algoritmos e Estruturas de Dados I

Universidade de São Paulo

Segundo Semestre de 2020

Structs

Definição de tipos

Ponteiros

Variáveis estruturadas - structs

Podemos declarar variáveis que armazenam valores de tipos diferentes: `structs`

```
struct aluno {
  char nome[MAX];
  long nusp;
  double nota;
} a1, a2;
```

As variáveis `a1` e `a2` são do tipo `struct aluno`. Para acessar os vários campos destas variáveis usamos o operador `.` (ponto):

```
a1.nusp = 11110976;
scanf ("%[^\n]", a2.nome);
a2.nota = 9.6;
```

Mais `structs`

- Faça uma `struct circ` que armazena as coordenadas do centro e o valor do raio de um círculo.
  1. Faça uma função que recebe um círculo e devolve o valor de sua área.
  2. Faça uma função que recebe um círculo e um fator de escala e devolve um círculo com o mesmo centro e o raio multiplicado por este escalar.

- Faça uma `struct complexo` que armazena um número complexo (parte real e imaginária). Escreva as funções que fazem operações sobre números complexos (soma, multiplicação, valor absoluto, etc).

Definição de tipos

A linguagem de programação C permite a definição de tipos novos, usando `typedef`.

```
typedef <tipo> <nome do tipo>;
```

```
typedef struct circ {
           double xc;
           double yc;
           double r;
         } circulo;
```

O tipo `circulo` é criado e podemos usá-lo em declarações:

```
circulo a, b;
```

Outro exemplo:

```
typedef int dia;
dia d1, d2;
```

Vetores e structs

- Escreva um tipo `ponto`, e uma função que devolve a distância entre dois pontos no plano.
- Escreva um programa que leia um ponto `origem` e $n > 0$ pontos no plano cartesiano e determina o ponto mais próximo da origem.

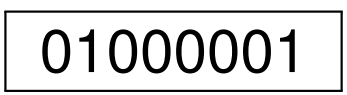
Endereços e ponteiros

A linguagem C permite manipular diretamente o endereço de memória das variáveis. Isso traz vantagens mas é também fonte comum de erros.

A memória (*stack*) é uma estutura linear, uma sequência de bytes. Os bytes podem estar sendo usado para armazenar variáveis de vários tipos (`int`, `double`, `char`, etc.). Estes bytes são numerados sequencialmente. Este é o endereço daquele byte na memória.

```
char c = 'A'; /* código ASCII = 65 */
```

`0xeea7cae7`



Operadores & e *

```
int x = -5;
int *p = &x;

printf ("x = %d, %d \n", x, *p);
*p = 10;
printf ("x = %d, %d \n", x, *p);

double a = 1234.5678;
double * pa = &a;
```

Qual a diferença?

```
p = p / 10;
*p = *p /10;
```