Algoritmos e Estruturas de Dados I

Estruturas

Pedro O.S. Vaz de Melo

Problema 1

- Implementar uma calculadora para fazer operações sobre frações (ex: 1/3, 5/13 etc). A calculadora deve ser capaz de realizar as seguintes operações:

  - somar
  - dividir
  - subtrair
  - multiplicar
  - simplificar

Análise do programa

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

void obter_fracoes();
void somar_fracoes(int x, int y, int u, int v);
void subtrair_fracoes(int x, int y, int u, int v);
void multiplicar_fracoes(int x, int y, int u, int v);
void dividir_fracoes(int x, int y, int u, int v);
void simplificar_fracao(int x, int y);

int a,b,c,d;

int main(int args, char * arg[])
{
  char r;

  while (1)
  {
    system("CLS");
    printf("1. Somar\n");
    printf("2. Subtrair\n");
    printf("3. Multiplicar\n");
    printf("4. Dividir\n");
    printf("9. Fim\n");
    printf("O que deseja? ");
    r = getche();
```

Análise do programa

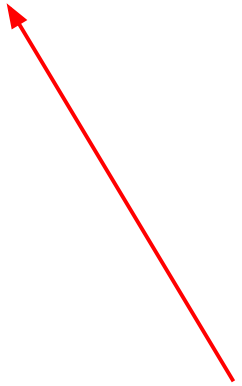
```c
#include <stdio.h>
#include <stdlib.h>

void obter_fracoes();
void somar_fracoes(int x, int y, int u, int v);
void subtrair_fracoes(int x, int y, int u, int v);
void multiplicar_fracoes(int x, int y, int u, int v);
void dividir_fracoes(int x, int y, int u, int v);
void simplificar_fracao(int x, int y);

int a,b,c,d;

int main(int args, char * arg[])
{
  char r;

  while (1)
  {
    system("CLS");
    printf("1. Somar\n");
    printf("2. Subtrair\n");
    printf("3. Multiplicar\n");
    printf("4. Dividir\n");
    printf("9. Fim\n");
    printf("O que deseja? ");
    r = getche();
```



confuso e com muitos parâmetros!

Análise do programa

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

typedef struct frac
{
  int num;
  int den;
} frac;

void obter_fracoes(frac *, frac *);
void somar_fracoes(frac, frac);
void subtrair_fracoes(frac, frac);
void multiplicar_fracoes(frac, frac);
void dividir_fracoes(frac, frac);
void simplificar_fracao(frac);

int main(int args, char * arg[])
{
  char r;
  frac a,b;

  while (1)
  {
    system("CLS");
    printf("1. Somar\n");
    printf("2. Subtrair\n");
    printf("3. Multiplicar\n");
    printf("4. Dividir\n");
    printf("9. Fim\n");
    printf("O que deseja? ");
    r = getche();
```

Define um novo tipo de dados. O tipo frac!

O novo tipo pode ser usado nos parâmetros das funções

Definição de novos tipos de dados

- Se cada fração compreende dois inteiros, como é possível fazer uma função para somar duas frações passando apenas dois parâmetros?

- Isto é possível porque a linguagem C permite a definição de novos tipos de dados com base nos tipos primitivos: char, int, float e double.

- Estes novos tipos de dados, formados a partir dos tipos primitivos são chamados de tipos estruturados.

Definição de novos tipos de dados

- Uma variável de um determinado tipo estruturado definido pelo usuário é comumente chamada de uma estrutura.
- Uma estrutura agrupa várias variáveis de diversos tipos em uma só variável.
- Para criar uma estrutura usa-se o comando struct:

```
struct nome_da_estrutura
{
    tipo_1 variavel_1;
    ...
    tipo_n variavel_n;
};
```

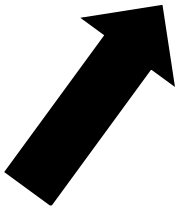
As variáveis que compõem a estrutura são chamadas de campos da estrutura.

Definição de novos tipos de dados

Exemplos:

```
struct ponto
{
    float coord_x;
    float coord_y;
};
```

```
struct cilindro
{
    float altura;
    struct circulo base;
};
```




```
struct circulo
{
    float raio;
    struct ponto centro;
};
```


A declaração de variáveis de um tipo estruturado (estruturas) é feita da mesma forma que para um tipo simples.

Definição de novos tipos de dados

- Para se acessar os campos de uma estrutura, basta separar o nome da variável pelo símbolo ponto (**.**).

- Para os exemplos anteriores:

```
struct ponto
{
    float coord_x;
    float coord_y;
};
```

```
struct cilindro
{
    float altura;
    struct circulo base;
};
```

```
struct circulo
{
    float raio;
    struct ponto centro;
};
```

```
struct cilindro d;
d.altura = 3.0;
d.base.raio = 5.5;
d.base.centro.coord_x = 1.2;
d.base.centro.coord_y = 3.8;
```

O comando typedef

- O Comando typedef permite ao programador definir um novo nome para um determinado tipo.

- Sua forma geral é:

```
typedef nome_antigo nome_novo;
```

- Exemplo:

Dando o nome inteiro para o tipo int:

```
typedef int inteiro;

inteiro num;
```

O comando typedef

- O comando typedef também pode ser utilizado para dar nome a tipos complexos como estruturas.

- Exemplos:

```
typedef struct tipo_endereco
        {
         char rua[50];
         int  numero;
         char bairro[20];
         char cidade[30];
         char sigla_estado[3];
         long int CEP;
        } TEndereco;
```

```
typedef struct frac
        {
         int num;
         int den;
        } frac;
```

O comando typedef

- Observação:
  Utilizando-se o comando struct juntamente com o comando typedef, pode-se dispensar o uso da palavra struct na declaração da variável.

- Exemplos:

```
typedef struct ponto
        {
          float x;
          float y;
        } ponto;
```

```
typedef struct circulo
        {
           float raio;
           ponto centro;
        } circulo;
```

```
typedef struct cilindro
        {
           float altura;
           circulo base;
        } cilindro;
```

Estruturas como parâmetros de funções

- Funciona como qualquer outro tipo de variável

```
typedef struct ponto
        {
          float x;
          float y;
        } ponto;
```

```
float distancia_pontos(ponto p1, ponto p2) {
   float parte1 = pow(p1.x - p2.x,2);
   float parte2 = pow(p1.y - p2.y,2);
   return sqrt(parte1 + parte2);
}
```

```
void main() {
   ponto u, v;
   scanf("%f %f %f %f", &u.x, &u.y, &v.x, &v.y);
   printf("\n %f", distancia_pontos(u,v));
}
```

Estruturas como parâmetros de funções

- E como faço para passar estruturas por referência?

```
void le_coordenada(ponto *p1) {
   float a, b;
   printf("Digite a coordenada x e y\n");
   scanf("%f %f", &a, &b);
   p1->x = a;
   p1->y = b;
}
```

```
void main() {
   ponto u, v;
   le_coordenada(&u);
   le_coordenada(&v);
   printf("\n %f", distancia_pontos(u,v));
   getch();
}
```

Estruturas como parâmetros de funções

```
void main() {
    ponto u, v;
    le_coordenada(&u);
    le_coordenada(&v);
    printf("\n %f", distancia_pontos(u,v));
    getch();
}
```

endereço	variável	conteúdo
F000	u | x	
F010	u | y	
F020	v | x	
F030	v | y	
F040		
F050		
F060		
F070		

Estruturas como parâmetros de funções

```
void le_coordenada(ponto *p1) {
    float x, y;
    printf("Digite a coordenada x e y\n");
    scanf("%f %f", &x, &y);
    p1->x = x;
    p1->y = y;
}
```

endereço	variável	conteúdo
F000	u | x	
F010	u | y	
F020	v | x	
F030	v | y	
F040	**p1**	**F000**
F050	**x**	**1.5**
F060	**y**	**2.0**
F070		

```
le_coordenada(&u);
```

Estruturas como parâmetros de funções

```
void le_coordenada(ponto *p1) {
    float x, y;
    printf("Digite a coordenada x e y\n");
    scanf("%f %f", &x, &y);
    p1->x = x;
    p1->y = y;
}
```

endereço	variável	conteúdo
F000	u | x	
F010	u | y	
F020	v | x	
F030	v | y	
F040	**p1**	**F000**
F050	**x**	**1.5**
F060	**y**	**2.0**
F070		

```
le_coordenada(&u);
```

Similar a vetores, **&u** dá o endereço inicial que a estrutura está armazenada

Estruturas como parâmetros de funções

```
void le_coordenada(ponto *p1) {
    float x, y;
    printf("Digite a coordenada x e y\n");
    scanf("%f %f", &x, &y);
    p1->x = x;
    p1->y = y;
}
```

endereço	variável	conteúdo
F000	**u	x**	**1.5**
F010	**u	y**	**2.0**
F020	v	x	
F030	v	y	
F040	p1	F000
F050	x	1.5
F060	y	2.0
F070		

```
le_coordenada(&u);
```

Estruturas como parâmetros de funções

```
void le_coordenada(ponto *p1) {
    float x, y;
    printf("Digite a coordenada x e y\n");
    scanf("%f %f", &x, &y);
    p1->x = x;
    p1->y = y;
}
```

endereço	variável	conteúdo
F000	u | x	1.5
F010	u | y	2.0
F020	**v | x**	**30.0**
F030	**v | y**	**25.5**
F040	p1	**F020**
F050	x	**30.0**
F060	y	**25.5**
F070		

```
le_coordenada(&v);
```

Estruturas como parâmetros de funções

```
void main() {
    ponto u, v;
    le_coordenada(&u);
    le_coordenada(&v);
    printf("\n %f", distancia_pontos(u,v));
    getch();
}
```

endereço	variável	conteúdo
F000	u | x	**1.5**
F010	u | y	**2.0**
F020	v | x	**30.0**
F030	v | y	**25.5**
F040		
F050		
F060		
F070		