USP-ICMC-BInfo

Estrutura, União e Enumeração em C

SCC501 - ICC-II

2011

Prof. João Luís

Introduzindo Estruturas

Estrutura: coleção de tipos diferentes.

Define-se primeiro o tipo:

```
struct inflavel
{
   char nome[20];
   float volume;
   double preco;
};
```

Depois as variáveis deste tipo:

```
struct inflavel chapeu;        // chapeu tem uma estrutura do
                               // tipo inflavel
struct inflavel mainframe;     // variável do tipo inflavel
struct inflavel ganso;
struct inflavel vincent;
```

```
#include <stdio.h>

main()
{
   struct inflavel
   {
        char nome[20];
        float volume;
        double preco;
   };

   struct inflavel bouquet =
              {
              "margarida",
              (float) 0.20,
              12.49
              };

   struct inflavel escolha;
```

```
    printf("bouquet: %s por R$", bouquet.nome);
    printf("%g\n", bouquet.preco);
    escolha = bouquet; // atribui uma estrutura a outra
    printf("escolha: %s por R$", escolha.nome);
    printf("%g\n", escolha.preco);
}
```

Saída:

```
bouquet: margarida por R$12.49
escolha: margarida por R$12.49
```

```
struct estac
{
   int nro_chave;
   char carro[12];
} sr_smith, sr_jones;          // duas variáveis estac

struct estac
{
   int nro_chave;
   char carro[12];
} sr_glitz =
{
   7,                  // valor para o membro sr_glitz.nro_chave
   "Packard"           // valor para o membro sr_glitz.carro
};

struct                 // sem "tag" (nome da estrutura)
{
   int x;              // 2 membros
   int y;
} posicao;             // uma variável estrutura
```

Vetores de Estruturas

```
struct inflavel presentes[100];   // vetor de 100
                                  // estruturas inflavel
scanf("%f", &(presentes[0].volume));// usa o membro volume
                                  // da primeira estrutura
printf("%g\n", presentes[99].preco); // mostra o membro
                                  // preço da última
                                  // estrutura

struct inflavel convidados[2] =
{
   {"Bambi", 0.5, 21.99},     // primeira estrutura no vetor
   {"Godzilla", 2000, 565.99} // segunda estrutura no vetor
};
```

Uniões

- Uma união é um formato de dados que pode armazenar tipos diferentes mas apenas um tipo de cada vez. Uma estrutura (**`struct`**) pode armazenar um **`int`** ***e*** um **`long`** ***e*** um **`double`**, uma união (**`union`**) pode armazenar um **`int`** ***ou*** um **`long`** ***ou*** um **`double`**:

```
union umpratodos;
{
  int val_int;
  long val_long;
  double val_double;
};
```

```
#include <stdio.h>
main()
{
   union umpratodos
   {
        int val_int;
        long val_long;
        double val_double;
   };

   struct inventario
   {
        char marca[20];
        union          // formato depende do tipo inventario
        {
               long nro_id;          // inventários do tipo 1
               char char_id[20];     // outros inventários
        } id;
        int tipo;
   };
```

```
    union umpratodos container;
    struct inventario preco;
    container.val_int = 15;                 // armazena um int
    printf("%d\n\n", container.val_int);
    container.val_double = 1.38;   // armazena um double, o
                                   // valor int é perdido
    printf("%g\n\n", container.val_double);
    printf("Qual o tipo? (1 para inteiro): ");
    scanf("%d", &(preco.tipo));
    if (preco.tipo == 1)
    {
        printf("Entre com o numero: ");
        scanf("%d", &(preco.id.nro_id)); // usa nome do
                              // membro para indicar modo
    }
    else
    {
        printf("Entre com o nome: ");
        scanf("%s", &(preco.id.char_id));
    }
}
```

União anônima

```
#include <stdio.h>

main()
{
   struct inventario
   {
        char marca[20];
        union         // formato depende do tipo inventario
        {
               long nro_id;         // inventários do tipo 1
               char char_id[20];    // outros inventários
        };
        int tipo;
   };

   struct inventario preco;
```

```
    printf("Qual o tipo? (1 para inteiro): ");
    scanf("%d", &(preco.tipo));

    if (preco.tipo == 1)
    {
        printf("Entre com o numero: ");
        scanf("%d", &(preco.nro_id)); // usa nome do membro
                                      // para indicar modo
    }
    else
    {
        printf("Entre com o nome: ");
        scanf("%s", &(preco.char_id));
    }
}
```

Enumerações

```
enum espectro {vermelho, laranja, amarelo,
  verde, azul, violeta, anil, ultravioleta};
```

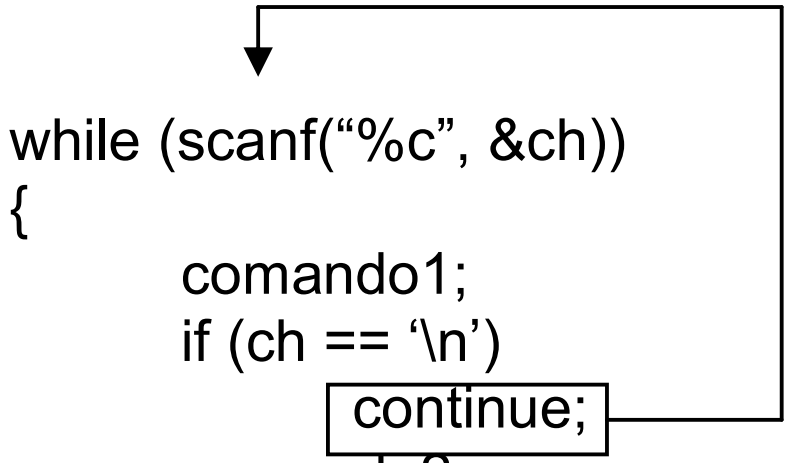
Este comando faz duas coisas:

1. Torna **`espectro`** o nome de um novo tipo; **`espectro`** é chamado de *enumeração*.
2. Estabelece **`vermelho`**, **`laranja`**, **`amarelo`**, etc. como constantes simbólicas para os inteiros 0-7. Estas constantes são chamadas de *enumeradores*.

```
enum espectro {vermelho, laranja, amarelo, verde,
azul, violeta, anil, ultravioleta};

enum espectro banda;

enum {xis, ipslon, ze};

enum bits {um = 1, dois = 2, quatro = 4, oito =
8};

enum grandepasso { primeiro, segundo = 100,
terceiro};

enum {zero, nulo = 0, hum, numero_um = 1};
```

```
int cor = azul; // válido, tipo espectro promovido a int

printf("%d\n", cor);

banda = azul; // válido, azul é um enumerador

printf("%d\n", banda);

banda = 2000; // aceita porém 2000 não é um enumerador

banda = laranja; // válido

++banda; // incrementa o inteiro correspondente

banda = laranja + amarelo;

printf("%d\n", banda);

banda = 3 + anil; // válido, vermelho é convertido para int

printf("%d\n", banda);

printf("\nEntre com um numero inteiro: ");
scanf("%d", &banda);
printf("%d\n", banda);
```

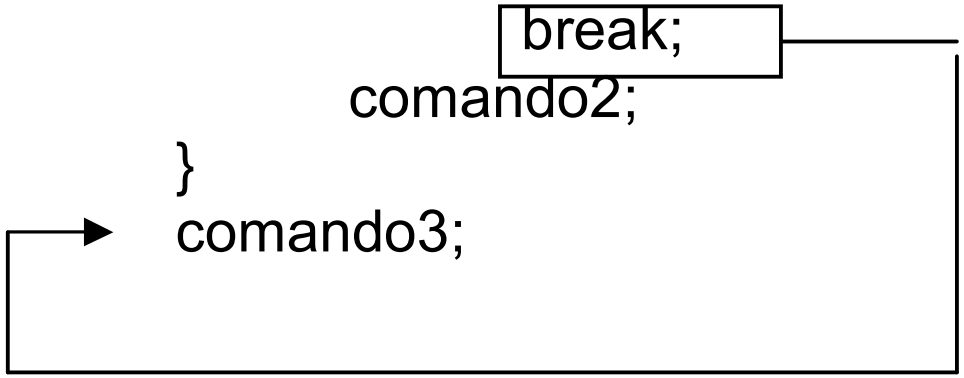
Os comandos **`break`** e **`continue`**



```
while (scanf("%c", &ch))
{
        comando1;
        if (ch == '\n')
                continue;
        comando2;
}
comando3;
```

continue pula o resto do corpo do loop e começa um novo ciclo.

```
while (scanf("%c", &ch))
{
        comando1;
        if (ch == '\n')
                break;
        comando2;
}
comando3;
```



break pula o resto do loop e vai ao comando seguinte.

```c
#include <stdio.h>

#define TamVet 80

main()
{
   char linha[TamVet];
   int i, espacos = 0;
   printf("Entre com uma linha de texto:\n");
   gets(linha);

   for (i = 0; linha[i] != '\0'; i++)
   {
       printf("%c", linha[i]); // mostra caractere
       if (linha[i] == '.')// sai se for um ponto
              break;
       if (linha[i] != ' ')// pula resto do loop
              continue;
       espacos++;
   }
   printf("\n%d espacos\n", espacos);
}
```

Saída:

```
Entre com uma linha de texto:
Vamos comer um lanche. Você paga!
Vamos comer um lanche.
3 espaços
```

Funções e Estruturas
Passando e Retornando Estruturas

```
#include <stdio.h>

struct hora_viagem {int horas; int mins;};
const int Mins_por_hr = 60;
struct hora_viagem soma(struct hora_viagem t1, struct hora_viagem
   t2);
void mostra_tempo(struct hora_viagem t);

main()
{
    struct hora_viagem dia1 = {5, 45};    // 5 hrs, 45 min
    struct hora_viagem dia2 = {4, 55};    // 4 hrs, 55 min
    struct hora_viagem dia3 = {4, 32};
    struct hora_viagem viagem = soma(dia1, dia2);
    printf("Total de dois dias: ");
    mostra_tempo(viagem);
    printf("Total de tres dias: ");
    mostra_tempo(soma(viagem, dia3));
}
```

```
struct hora_viagem soma(struct hora_viagem t1, struct
   hora_viagem t2)
{
    struct hora_viagem total;
    total.mins = (t1.mins + t2.mins) % Mins_por_hr;
    total.horas = t1.horas + t2.horas +
                        (t1.mins + t2.mins) / Mins_por_hr;
    return total;
}

void mostra_tempo(struct hora_viagem t)
{
    printf("%d horas, %d minutos\n", t.horas, t.mins);
}
```

Saída:

```
Total de dois dias: 10 horas, 40 minutos
Total de tres dias: 15 horas, 12 minutos
```

Outro exemplo

```
#include <stdio.h>
#include <math.h>
// definições das estruturas
struct Polar
{
   float distancia;// distância da origem
   float angulo;   // direção da origem
};
struct ret
{
    float x;        // distância horizontal da origem
    float y;        // distância vertical da origem
};

// protótipos
struct Polar ret_p_polar(struct ret xypos);
void mostra_polar(struct Polar dapos);
```

```
main()
{
    struct ret rlugar;
    struct Polar plugar;

    printf("Entre com os valores de x e y: ");
    while (scanf("%f%f", &(rlugar.x), &(rlugar.y)))
    {
          plugar = ret_p_polar(rlugar);
          mostra_polar(plugar);
          printf("Proximos dois numeros (s para sair): ");
    }
}

// converte coordenadas retangulares para polares
struct Polar ret_p_polar(struct ret xypos)
{
    struct Polar resp;

    resp.distancia =
          (float) sqrt( xypos.x * xypos.x + xypos.y * xypos.y);
    resp.angulo = (float) atan2(xypos.y, xypos.x);
    return resp;      // retorna uma estrutura Polar
}
```

```
// mostra coordenadas polares, convertendo ângulo em graus
void mostra_polar (struct Polar dapos)
{
    const double Rad_p_grau = 57.29577951;

    printf("distancia = %g", dapos.distancia);
    printf(", angulo = %g", dapos.angulo * Rad_p_grau);
    printf(" graus\n");
}
```

Saída:

```
Entre com os valores de x e y: 30 40
distancia = 50, angulo = 53.1301 graus
Proximos dois numeros (s para sair): -100 100
distancia = 141.421, angulo = 135 graus
Proximos dois numeros (s para sair): s
```

Passando endereços de estruturas

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

// definições de estruturas
struct Polar
{
    float distancia;    // distância da origem
    float angulo;       // direção da origem
};
struct ret
{
    float x;         // distância horizontal da origem
    float y;         // distância vertical da origem
};

// protótipos
void ret_p_polar(const struct ret * pxy, struct Polar *
   pda);
void mostra_polar (const struct Polar * pda);
```

```
main()
{
    struct ret rlugar;
    struct Polar plugar;

    printf("Entre com os valores de x e y: ");
    while (scanf("%f%f", &(rlugar.x), &(rlugar.y)))
    {
          ret_p_polar(&rlugar, &plugar);    // passa endereços
          mostra_polar(&plugar);         // passa endereço
          printf("Proximos dois numeros (s para sair): ");
    }
}

// converte coordenadas retangulares para polares
void ret_p_polar(const struct ret * pxy, struct Polar * pda)
{
    pda->distancia =
          (float) sqrt(pxy->x * pxy->x + pxy->y * pxy->y);
    pda->angulo = (float) atan2(pxy->y, pxy->x);
}
```

```
// mostra coordenadas polares, convertendo ângulo para graus
void mostra_polar (const struct Polar * pda)
{
    const double Rad_p_grau = 57.29577951;

    printf("distancia = %g", pda->distancia);
    printf(", angulo = %g", pda->angulo * Rad_p_grau);
    printf(" graus\n");
}
```

Saída:

```
Entre com os valores de x e y: 10 -20
distancia = 22.3607, angulo = -63.435 graus
Proximos dois numeros (s para sair): 0 100
distancia = 100, angulo = 90 graus
Proximos dois numeros (s para sair): s
```