Aula 22

MC 102 - Algoritmos e Programação de Computadores

Definição de tipos, Registros e Tipos Enumerados.

Definição de Tipos

Algumas vezes, é interessante declarar um tipo de dado próprio, para organização do código, que representa exatamente outro tipo de dado existente.

Por exemplo: seria interessante definir um tipo *nota* para armazenar os valores das notas de alunos, ou o tipo *Idade*, para armazenar a idade dos alunos.

Mas como isso pode ser feito?

Definindo Tipos: comando typedef

typedef <tipo_existente> <novo_tipo>;

Obs: Os tipos devem ser declarados antes da função main

Exemplos:

```
typedef unsigned int uint;
typedef char letra;
typedef float nota;
```

Usando Tipos Redefinidos

```
#include <stdio.h>
typedef unsigned int uint;
typedef char letra;
typedef float nota;
int main (int argc, char **argv) {
  uint idade;
  nota P1;
  letra turma;

  printf ("Digite sua idade: ");
  scanf ("%u", &idade);
  printf ("Digite sua turma: ");
  scanf ("%c", &turma);
  printf ("Digite sua nota P1: ");
  scanf ("%f", &P1);
  ...
}
```

A utilização dos novos tipos definidos é igual a dos tipos primários.

Registros

Usando um vetor, podemos definir uma variável para um conjunto de dados de mesmo tipo. Mas como usar um conjunto de dados de tipos diferentes?

Um Registro é uma variável que contém diversas variáveis (chamadas de campos) que podem ser de tipo de diferentes.

Por exemplo, podemos definir uma variável Aluno para armazenar o RA, o Nome e sua Turma.

Declaração de Registros: **struct**

```
struct <nome_da_strutura> {
   <tipo_de_dado1> <nome_1>;
   <tipo_de_dado2> <nome_2>;
};
```

Obs: Os registros também devem ser declarados antes da função main

Exemplos:

```
struct RegAluno {
   unsigned int RA;
   char Nome[30];
};
```

Definindo os Registros

```
struct RegAluno {
   unsigned int RA;
   char Nome[30];
};
....
struct RegAluno Alunos;


struct {
   unsigned int RA;
   char Nome[30];
} AlunosTurmaIJ, AlunosTurmaKL;
```

Acesso aos Membros do Registro

Acesso aos membros:

<nome_var>.<nome_membro>

Exemplo:

```
struct RegAluno {
   unsigned int RA;
   char Nome[30];
};
....
struct RegAluno Alunos;

Alunos.RA = 12345;
strcpy(Alunos.Nome,"Joao");
```

Lembre-se: para atribuir um valor para uma string, deve-se usar sempre o **strcpy** e **nunca o operador =**

Usando Registros

```
#include <stdio.h>
struct RegAluno {
   unsigned int RA;
   char Nome[30];
};

int main (int argc, char **argv) {
   struct RegAluno Aluno;

   printf ("Digite seu RA: ");
   scanf ("%d", &(Aluno.RA));
   printf ("Digite seu nome: ");
   gets (Aluno.Nome);
   ...
}
```

Para acessar um campo, é necessário usar "." entre o nome da variável e o nome do campo.

A palavra-chave `struct` sempre deve ser usada antes do nome do registro.

Definindo o Tipo Registro

`typedef struct` – Tipo Registro

```
typedef struct <nome_da_strutura> {
   <tipo_de_dado1> <nome_1>;
   <tipo_de_dado2> <nome_2>;
} <novo_tipo>;
```

Obs: Os registros também devem ser declarados antes da função main

Exemplos:

```
typedef struct RegistroAluno {
   unsigned int RA;
   char Nome[30];
} RegAluno;
```

Usando Tipo Registro

```
#include <stdio.h>
typedef struct RegistroAluno {
  unsigned int RA;
  char Nome[30];
} RegAluno;

int main (int argc, char **argv) {
  RegAluno Aluno;

  printf ("Digite seu RA: ");
  scanf ("%d", &(Aluno.RA));
  printf ("Digite seu nome: ");
  gets (Aluno.Nome);
  ...
}
```

Note que a palavra-chave `struct` não é mais necessária pois o registro agora está associado a um tipo de dado definido.

Registros Aninhados

Constantemente precisamos definir Registros em que um ou mais campos também são registros.

Exemplo:

```
typedef struct RData {
  unsigned int Dia;
  unsigned int Mes;
  unsigned int Ano;
} Data;
typedef struct RPeriodo {
  Data Inicio;
  Data Fim;
} Periodo;
```

Vetores de Registros

```
#include <stdio.h>
typedef struct RegistroAluno {
  unsigned int RA;
  char Nome[30];
} RegAluno;

int main (int argc, char **argv) {
  RegAluno Alunos[10];

  strcpy(Alunos[0].Nome, "Fulano");
  Alunos[0].RA = 79591;
  ...
  /* Copiando Registros */
  Alunos[3] = Alunos[0];
}
```

Para se copiar os valores de um registro, pode-se copiar campo por campo ou apenas atribuir um registro inteiro a outra variável do mesmo tipo, de forma direta.

Tipos Enumerados

Em C há um tipo especial usado para declarar tipos enumerados. Ele serve para definir um conjunto finito de elementos que podem ser usados.

Exemplo:

sexo: masculino, feminino

dia_util: segunda, terça, quarta, quinta, sexta

Tipos enumerados são úteis para auxiliar a programação e dar um significado semântico para algumas variáveis.

Tipos Enumerados

Declaração:

```
enum <tipo> {valor1,valor2,valor3,...};
```

Exemplos:

```
enum tiposexo {masculino,feminino};
enum booleano {falso,verdadeiro};
```

Os valores são indexados como inteiros, de 0 até n-1, para n valores enumerados.

Tipos Enumerados

Declaração:

```
enum <tipo> {valor1,valor2,valor3,...};
```

Exemplos:

```
enum tiposexo {masculino,feminino};
enum booleano {falso,verdadeiro};
```

Os valores são indexados como inteiros, de 0 até n-1, para n valores enumerados.

Usando Tipos Enumerados

```
#include <stdio.h>
enum tiposexo {masculino,feminino};
enum booleano {falso,verdadeiro};

int main (int argc, char **argv) {
  tiposexo sexo;
  booleano continua;

  continua = falso;
  sexo = masculino;
  ...
  if (continua==verdadeiro) {
    if (sexo==feminino)
      ...
  ...
  sexo = 1;
}
```

As variáveis de tipos enumerados são semelhantes a variáveis inteiras. Tanto um dos valores enumerados quanto um inteiro podem ser atribuídos a elas.

Tipos Enumerados

Os índices associados a um valor enumerado podem ser definidos para cada valor. Se um valor não possui definição de índice, seu índice será o anterior mais um.

Exemplos:

```
enum meses {jan=1,fev,mar,abr,mai,jun,jul,
            ago,set,out,nov,dez};
enum coordenadas {leste,norte=90,
                  oeste=180,sul=270};
```

Usando Tipos Enumerados

```
#include <stdio.h>
enum meses {jan=1,fev,mar,abr,mai,jun,
            jul,ago,set,out,nov,dez};

int main (int argc, char **argv) {
  meses Mes;

  Mes = jan;
  ...
  if (Mes<=jul) {
    Mes = Mes + 3;
  else
    Mes = Mes - 1 ;
  ...
}
```

Operadores matemáticos para inteiros podem ser utilizados também em tipos enumerados.

Resumo

- Definição de tipos e tipos enumerados facilitam a programação e permitem melhorar o entendimento do programa.
- Estruturas são muito úteis para definir um conjunto de campos de valores para diferentes tipos de dados que podem ser armazenados em uma variável.