MC-102 — Aula 20
Redefinição de tipos, Constantes, Registros

Instituto de Computação – Unicamp

Segundo Semestre de 2011

Roteiro

Redefinido um tipo

- Às vezes, por questão de organização, gostaríamos de criar um tipo próprio nosso, que faz exatamente a mesma coisa que um outro tipo já existente.
- Isso é util quando desenvolvemos programas grandes onde a alteração do tipo de uma determinada variável para outra acarretaria na alteração de muitas variáveis.
- Por exemplo, em um programa onde manipulamos médias de alunos, todas as variáveis que trabalhassem com nota tivessem o tipo `nota`, e não `int` ou `float`.

O comando typedef

- A forma de se fazer isso é utilizando o comando `typedef`, seguindo a estrutura abaixo:

```
typedef <tipo_ja_existente> <tipo_novo>;
```

- Usualmente, fazemos essa declaração fora da função `main()`, embora seja permitido fazer dentro da função também.
- Ex: `typedef float nota;`
  Cria um novo tipo, chamado nota, cujas variáveis desse tipo serão pontos flutuantes.

Exemplo de uso do typedef

```
#include <stdio.h>
typedef float nota;
main () {
  nota P1;
  printf ("Digite a nota 1\n");
  scanf ("%f", &P1);
  printf ("A nota 1 foi %f\n", P1);
}
```

Veja mais detalhes em `typedef.c`

Constantes

Frequentemente, utilizamos um determinado valor diversas vezes durante um programa. Para ilustrar, vamos imaginar um programa que trabalhe com um vetor ou uma matriz.

- Na declaração de um vetor limitamos o seu tamanho a um determinado tamanho máximo;

Constantes

- Na leitura e na escrita é comum lermos todos os elementos do vetor;
- Em geral percorremos o vetor uma ou mais vezes durante o nosso programa.

Em todos esses casos, utilizamos um mesmo valor para limitar o laço `for` que percorrerá todo o vetor (ou matriz) e para a declaração.

Constantes

- Uma forma de utilizarmos uma única representação para esse valor é declarar uma variável e atribuir um valor constante a ela, mas isso só resolve o problema dos laços, não o da declaração do vetor.

Constantes

- Para resolver isso, utilizamos o comando `#define`, que permite definir constantes dentro do programa que podem ser utilizadas em qualquer lugar onde uma constante seria utilizada (inclusive na declaração de vetores).

  Ex: `#define MAX_ELEMENTOS 10`

  define a constante `MAX_ELEMENTOS` com o valor 10

Constantes

- O formato padrão do comando `#define` é

```
#define NOME_DA_CONSTANTE <valor_da_constante>
```

- Qualquer tipo de constante pode ser colocada no lugar da constante do `#define`, como pontos flutuantes, cadeias de caracteres, etc...

Constantes

- O comando `#define` deve ser utilizado SEMPRE abaixo do comando `#include` inicial ou de outro comando define, nunca dentro do `main ()`.
- Tipicamente utilizamos letras maiúsculas para o nome das constantes e letras minúsculas para o resto do programa (nome de variáveis, comandos, etc ...)

Constantes

- Uma constante não tem tipo. Na verdade, o compilador verifica todos os lugares onde você usou a constante e substitui pelo valor à direita ANTES de realizar a compilação. Esse processo é chamado de pré-compilação.
- Uma boa forma de organizar o seu programa é colocar constantes no começo dele, com nomes claros. Logo ao abrir o seu código será possível identificar os limites de seu programa.

Exemplo de uso do #define

```
#include <stdio.h>
#define NOTA 10
#define MENSAGEM "Parabens, nota %d\n"
main () {
  printf ("Parabens, nota %d\n", 10);
  printf ("Parabens, nota %d\n", NOTA);
  printf (MENSAGEM, NOTA);
}
```

Veja o programa define.c
(teste gcc -E para ver a pré-compilação)

Armazenando vários dados

Como armazenar os dados(RA, telefone, nome, endereço e data de nascimento) de um aluno?

```
int ra;
int telefone;
char nome[30];
char endereco[100];
int dia_nascimento;
int mes_nascimento;
int ano_nascimento;

scanf("%d%d%s%s%d%d%d", &ra, &telefone, nome, endereco,
 &dia_nascimento, &mes_nascimento, &ano_nascimento);
```

Copiando Dados

E se quiséssemos copiar os dados do aluno?

```
int ra, copia_ra;
int telefone, copia_telefone;
char nome[30], copia_nome[30];
char endereco[100], copia_endereco;
int dia_nascimento, copia_dia_nascimento;
int mes_nascimento, copia_mes_nascimento;
int ano_nascimento, copia_ano_nascimento;
int i;

copia_ra = ra;
copia_telefone = telefone;

for (i = 0; nome[i] != '\0'; i++)
copia_nome[i] = nome[i];
```

Passando dados por parâmetros

Ou passar os dados como parâmetros da função

```
void imprimir_dados(int ra, int telefone, char nome[],
char endereco[], int dia_nascimento,
int mes_nascimento, int ano_nascimento) {
...
}

int main() {
...
imprimir_dados(ra, telefone, nome, endereco,
dia_nascimento, mes_nascimento, ano_nascimento);
...
}
```

Registros

Um registro é uma variável que contém diversas variáveis (chamadas de campos), usualmente de tipos diferentes, mas que dentro de um determinado contexto fazem sentido se agrupadas. Podemos comparar um registro com uma ficha que possui todos os dados sobre uma determinada entidade, por exemplo:

- Registro de alunos (nome, RA, médias de provas, médias de labs, etc...)
- Registro de pacientes (Nome, endereço, histórico de doenças, etc...)

Declarando o formato do registro

A primeira parte da criação de um registro é declarar seu formato. Isso é feito utilizando a palavra chave `struct`, da seguinte forma:

```
struct nome_do_tipo_do_registro {
  tipo_1 nome_1;
  tipo_2 nome_2;
  tipo_3 nome_3;
  ...
  tipo_n nome_n;
};
```

Declarando o formato do registro

- A declaração do formato de uma estrutura pode ser feita dentro do seu programa (ou seja, na função `main`) ou fora dela. Usualmente, ela é feita fora da função, como no exemplo abaixo:

```
#include <stdio.h>

  /* Declare o formato de seu registro aqui */

main () {
  /* Construa seu programa aqui */
}
```

Declarando um registro

A próxima etapa é declarar uma variável do tipo `struct nome_do_tipo_da_estrutura`, que será usada dentro de seu programa, como no exemplo abaixo:

```
#include <stdio.h>

struct ficha { int ra; float media; };

main () {
  struct ficha f;
}
```

Utilizando os campos de um registro

Podemos acessar individualmente os campos de um determinado registro como se fossem variáveis normais, utilizando a seguinte estrutura

```
nome_do_registro.nome_do_campo
```

Utilizando os campos de um registro

- Para o registro declarado anteriomente, utilizaríamos

  ```
  f.ra
  ```

  para acessar o campo ra do registro f (note que usamos o nome da variável e não o nome dado ao formato do registro).
- Podemos colocar o campo de um registro em qualquer lugar onde colocaríamos uma variável.

Lendo os campos de um registro

A leitura dos campos de um registro a partir do teclado deve ser feita campo a campo, como se fossem variáveis independentes.

```
printf ("Digite o ra do aluno: ");
scanf ("%d", &f.ra);

printf ("Digite a média do aluno: ");
scanf ("%f", &f.media);
```

Escrevendo os campos de um registro

A escrita na tela do valor dos campos de um registro deve ser feita campo a campo, como se fossem variáveis independentes.

```
printf ("O aluno %d tirou média %f\n",
        f.ra, f.media);
```

Veja o exemplo em `leitura_escrita.c`

Copiando registros

A cópia de um registro pode ser feita como se fosse a cópia de uma variável normal, ou seja

```
registro_1 = registro_2
```

Veja o exemplo em `copia.c`

Vetor de registros

Pode ser declarado quando necessitarmos de diversas cópias de um registro (por exemplo, para cadastrar todos os alunos de uma mesma turma).

- Para declarar: `struct ficha f[5];`
- Para usar: `f[indice].campo;`

Veja o exemplo em `vetor.c`

Registros aninhados

Pode-se também declarar um registro como uma das variáveis de um registro, quantas vezes isso for necessário.

```
#include <stdio.h>
struct medias {
  float p1;
  float p2;
  float p3;
};
```

Registros aninhados

```
typedef struct medias Medias;
struct ficha {
  int ra;
  Medias provas;
};
```

Veja o exemplo em `aninhado.c`

Exercício

- Faça um programa para armazenar 20 nomes e 20 telefones em um vetor de registros, colocar os elementos do vetor em ordem crescente de nome, e depois imprimir o vetor ordenado.
- Escreva um programa para armazenar 10 produtos, o código, o preço unitário e a quantidade. O programa deverá ter 4 opções básicas: 1. Cadastrar Produto; 2. Buscar Produto (pelo código); 3. Imprimir Lista de Produtos; 4. Sair.