unipampa

Universidade Federal do Pampa

Laboratório de Programação I

Assunto de Hoje
Estruturas Compostas (structs)

Professor Luciano Brum
lucianobrum@unipampa.edu.br

unipampa
Universidade Federal do Pampa

Roteiro

- Conceitos Básicos.
- Declaração e inicialização de tipos estruturados.
- Tipos estruturados e funções.
- Operador typedef.
- Estruturas aninhadas.

Conceitos Básicos

- Até o momento no curso, foram trabalhados os tipos básicos fornecidos pela linguagem de programação C.

- Relembrando: char, float, int, double, etc.

Conceitos Básicos

- São necessários outros tipos e uma maior abstração conforme necessitarmos de programas mais complexos.

Conceitos Básicos

- Exemplo:
  - Um programa que trata informações de alunos. Quais informações poderiam ser necessárias para um aluno?
  - Nome, data de nascimento, matrícula, etc.

Conceitos Básicos

- É necessário que a linguagem ofereça um mecanismo para tratar o aluno como um objeto único (idade, nome e outros campos em uma mesma estrutura).

Conceitos Básicos

- A linguagem C oferece mecanismos para estruturas dados complexos para informações que contenham diversos campos.

- Podemos utilizar uma estrutura composta por tipos primitivos, outras estruturas e, ainda, ponteiro para estruturas (Foco em ED).

Conceitos Básicos

- Uma estrutura (struct) é uma coleção de variáveis referenciadas por um nome.
- Útil quando se deseja agrupar informações ("registros").
- As variáveis que formam a estrutura são chamados membros (ou campos ou elementos ou registros).
- Geralmente, todos os componentes de uma estrutura estão relacionados.

Conceitos Básicos

- Structs são Estruturas de Dados Heterogêneas.
- Uma struct pode agrupar várias variáveis numa só.
- A struct serve para agrupar um conjunto de dados não similares, formando um novo tipo de dados.

unipampa
Universidade Federal do Pampa

Roteiro

- Conceitos Básicos.
- Definição e Declaração de tipos estruturados.
- Tipos estruturados e funções.
- Operador typedef.
- Estruturas aninhadas.

Definição de Structs

- Exemplo: plano cartesiano. No plano cartesiano, temos os componentes x e y para um ponto no espaço 2D.

- Como representamos esta informação sem estruturas compostas?

Definição de Structs

- Sem estruturas, é possível que o programador confunda coordenadas x e y de pontos diferentes.

- Neste caso, vamos criar uma estrutura para pontos em um plano cartesiano.

Definição de Structs

Nome da struct

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

Componentes da struct 'ponto'

Definição de Structs

```
struct end {
    char nome[30];
    char rua[40];
    char cidade[20];
    char estado[3];
    unsigned long int cep;
};
```

Podemos ter vetores e matrizes em uma struct

Definição de Structs

- Quando uma variável de estrutura é declarada, o compilador C aloca automaticamente memória suficiente para acomodar todos os seus membros.
- Exemplo (assumindo caracteres com 1 byte e inteiros longos com 4 bytes):
  - nome (30 bytes)
  - rua (40 bytes)
  - cidade (20 bytes)
  - estado (3 bytes)
  - cep (4 bytes)

Declaração de Structs

Como é declarada uma Struct no programa principal?

Declaração de Structs

```
#include<stdio.h:
struct ponto{
        float x;
        float y;
};
...
int main(){
        struct ponto p;
        struct ponto p2[10];
}
```

O ‘p’ é uma variável do tipo ‘struct ponto’.

O ‘p2’ é uma variável do tipo vetor de ‘struct ponto’.

Declaração de Structs

Podemos declarar uma struct também no programa principal:

Declaração de Structs

```
struct data
{
int dia, mes, ano;
}    data1, data2;

data1.dia = 23;
data1.mes = 11;
data1.ano = 1971;
data2.dia = 15;
data2.mes = 9;
data2.ano = 2008;
```

‘data1’ e ‘data2’ são as variáveis do tipo ‘struct data’.

Exemplos de inicialização das structs.

Inicializando e acessando Structs

- Para acessar um componente de uma struct, utilizamos o operador de acesso “ponto” (.).

- Para inicializar um ponto seguindo o exemplo anterior:
  - p.x = 10.0;
  - p.y = 5.7;

Inicializando e acessando Structs

- Manipulamos os componentes de uma estrutura da mesma forma que variáveis simples.

- Podemos acessar seus valores, atribuir novos valores, acessar seus endereços de memória, etc.

Inicializando e acessando Structs

- Para ler as coordenadas de um ponto:

  - printf(“Digite as coordenadas de um ponto (x,y).”);
  - scanf(“%f %f ”, &p.x, &p.y);

Inicializando e acessando Structs

- Para imprimir as coordenadas de um ponto:

  - printf(“As coordenadas do ponto (x,y) informado são: (%f, %f).”, p.x, p.y);

unipampa
Universidade Federal do Pampa

Roteiro

- Conceitos Básicos.
- Declaração e inicialização de tipos estruturados.
- Tipos estruturados e funções.
- Operador typedef.
- Estruturas aninhadas.

Funções e Structs

- Podemos ter funções que recebem como parâmetro um ou mais tipos estruturados.

- Também podemos ter como retorno de uma função, um tipo estruturado.

Funções e Structs

- Exemplificando, a seguir duas funções, uma de leitura de pontos e outra para impressão dos pontos.

Funções e Structs

```
struct Ponto criaPonto(){
        struct Ponto s;
        printf("Digite as coordenadas de um ponto qualquer
(x,y).\n");
        scanf("%f%f ",&s.x,&s.y);
        return s;
}
void imprime(struct Ponto p){
        printf("O ponto fornecido foi: (%f,%f).\n", p.x, p.y);
}
```

Tipo de retorno da função.

Retorno da função.

Struct como parâmetro de função

No main:
struct ponto p1 = criaPonto();
Imprime(p1);

Funções e Structs

- Baseado nos exemplos anteriores é preciso ressaltar alguns pontos:

1. Ao passar a estrutura como argumento, faz-se uma cópia dela inteira para a pilha na memória e a função acessa os dados da cópia.

2. Na passagem de parâmetros por valor, a função não tem como alterar os valores dos componentes da estrutura principal.

3. Copiar uma struct inteira para a memória pode ser uma operação custosa e ineficiente.

Funções e Structs

- Solução eficiente: enviar para as funções o endereço da estrutura.

```
void captura(struct ponto* pp){
    printf(“Digite as coordenadas do ponto (x,y).\n”);
    scanf(“%f%f ”,&pp->x, &pp->y);
}
No main:
captura(&p);
```

Funções e Structs

- Por que o uso de ponteiros é mais eficiente?

- O tamanho de um ponteiro (endereço de uma variável ou estrutura) é geralmente de 4 bytes.

- O tamanho de uma estrutura pode ser imensamente maior dependendo da aplicação !

unipampa
Universidade Federal do Pampa

Roteiro

- Conceitos Básicos.
- Declaração e inicialização de tipos estruturados.
- Tipos estruturados e funções.
- Operador typedef.
- Estruturas aninhadas.

Typedef

- O que é o Typedef?
  - A linguagem C permite criar nomes de tipos com este operador.
  - Se escrevermos: typedef float Real;, poderemos utilizar o mnemônico Real para lidar com o tipo float.

Typedef

- Seguem outros exemplos válidos do uso do operador typedef:
  - typedef int* Pint;
  - typedef unsigned int Uint;
  - typedef struct ponto Ponto;

Typedef

- O typedef é muito utilizado com structs. Segue um exemplo:

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

OU

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

Typedef

- O que muda ao referenciar a estrutura? Podemos utilizar o novo nome definido no typedef.
- No main, a criação de uma estrutura fica:
  - Ponto p1;
- Em vez de:
  - struct ponto p1;
- O mesmo vale para passagem de parâmetros de funções, retornos, etc.

unipampa
Universidade Federal do Pampa

Roteiro

- Conceitos Básicos.
- Declaração e inicialização de tipos estruturados.
- Tipos estruturados e funções.
- Operador typedef.
- Estruturas aninhadas e vetores.

Aninhamento de Estruturas

- Uma estrutura pode ser o atributo de uma outra estrutura na linguagem C.

- Esta é a definição de um aninhamento de estruturas.

Aninhamento de Estruturas

- Exemplo: Quais podem ser as informações relevantes para o cadastro de um aluno?
- Nome, endereço, cidade, estado, país, nota do enem, curso, etc.
- Todos podem ser atributos ou elementos de uma struct do tipo Aluno.

Aninhamento de Estruturas

```
struct aluno{
    Nome[20];
    Endereço[20];
    ...
};
```

Aninhamento de Estruturas

```
struct aluno{
    Nome[20];
    Endereço[50];
    ...
};
```

- E se for desejada a pesquisa por todas as pessoas que moram na rua Júlio de Castilhos?

Aninhamento de Estruturas

```
struct aluno{
    char nome[20];
    struct endereço;
    ...
};
```

```
struct endereço{
        char rua[20];
        int numero;
        int cep;
        int bairro;
};
```

Aninhamento de Estruturas

- Exercício: crie um programa que informe na tela se um ponto pertence ou não à um círculo. O usuário deve informar os parâmetros do círculo e as coordenadas do ponto.

- Requisitos: use as estruturas Círculo e Ponto.
- Elementos de um círculo: raio e o centro do círculo (um ponto com coordenada x e y).
- Elementos de um ponto: coordenadas x e y.

Aninhamento de Estruturas

- Utilizem a biblioteca math.h para o uso da função sqrt.

- Função que calcula distância entre dois pontos:

Float distancia(Ponto p, Ponto q){

return $\sqrt{(x_2 - x_1)^2 + (y_2 - y_1)^2}$

}

Aninhamento de Estruturas

- Como definir se um ponto está no círculo?
  - Sabemos o raio do círculo e o ponto no centro.
  - Sabemos as coordenadas do ponto.
  - Se a distância entre o ponto e o centro do círculo é menor que o raio, é porque o ponto está dentro do círculo.

Vetores de Estruturas

- Podemos utilizar vetores para agrupar elementos dos tipos básicos, conforme visto em Algoritmos e Programação.

Vetores de Estruturas

- Podemos utilizar vetores para agrupar elementos dos tipos básicos, conforme visto em Algoritmos e Programação.

- Podemos também utilizar vetores para agrupar estruturas.

Vetores de Estruturas

- Exemplo prático: calcular o centro geométrico de um conjunto de 5 pontos. Considere que o CG também é um ponto. Mostre o CG no final do programa.

- Como calcular?

$$Cgx = \frac{\sum xi}{n} \qquad Cyx = \frac{\sum yi}{n}$$

Vetores de Estruturas

- Sugestão de função que calcule o CG:

```
Ponto centro_geom(int n, Ponto* v){
...
}
```

- No main:

```
Ponto p[5];
...
Ponto cg = centro_geom(numero_de_pontos, p);
```

Vetores de Estruturas

- Podemos passar para argumento de funções estruturas, partes de estruturas e vetores ou matrizes de estruturas.

- Podemos ter como retorno das funções uma estrutura ou o campo de uma estrutura.

- Cuidados: O tipo de retorno de função deve estar de acordo com o tipo retornado. A variável que recebe este retorno no programa principal também deve ser do mesmo tipo.

Vetores de Estruturas

- Estes recursos facilitam o uso das funções, porém, para estruturas muito grandes, devemos usar com critério este recurso.

- A cópia do valor de retorno pode ser caro computacionalmente.

Vetores de Estruturas

```
Struct aluno{
    Int mat;
    Char nome[81];
    Char end[121];
    Char tel[21];
};
```

```
Aluno tab[100];
```

Quanto custa uma estrutura aluno?

4+81+121+21 = 227 bytes

227 bytes * 100 = 22700
Bytes = 23KB

Vetores de Estruturas

- Ineficiente, pois não estaremos usando armazenando, de fato, 100 alunos.

- Estaremos desperdiçando memória.

Vetores de Estruturas

- Ineficiente, pois não estaremos usando armazenando, de fato, 100 alunos.

- Estaremos desperdiçando memória.

- Na próxima aula vamos ver como contornar e resolver esse problema.

unipampa

Universidade Federal do Pampa

Dúvidas ?