SSC0101 - ICC1 – Teórica

Introdução à Ciência da Computação I

Registros/Estruturas e Conjuntos

Prof. Vanderlei Bonato: vbonato@icmc.usp.br
Prof. Claudio Fabiano Motta Toledo: claudio@icmc.usp.br

Sumário

- Registros/Estruturas
- Declarando structs em linguagem C
- Acessando membros da estrutura
- Estrutura com funções
- Inicializando estruturas
- Exemplo de pilha
- Unions

Registros/Estruturas

- Coleção de variáveis geralmente de tipos diferentes.
- As variáveis são agrupadas usando um mesmo identificador com o objetivo de facilitar sua manipulação.
- O nome registro costuma ser empregado na linguagem Pascal.
- Estrutura (struct) é o nome utilizado em linguagem C.
- Registros/Estruturas permitem que um grupo de variáveis relacionadas sejam tratadas como uma unidade ao invés de entidades separadas.

Declarando structs em linguagem C

- Declaração de estruturas em linguagem C:

```
struct <etiqueta_estrutura>{
    <tipo_1>  <var_ou_estrutura_dados_1>;
    <tipo_2>  <var_ou_estrutura_dados_2>;
                  .....
    <tipo_N>  <var_ou_estrutura_dados_N>;
};
```

- Exemplos:

```
struct frutas
{
  char nome[10];
  int calorias;
};
```

```
struct complex
{
  float re;
  float im;
};
```

Declarando structs em linguagem C

- Declarando, iniciando e acessando variáveis usando struct.

```
struct complex
{
  float re;
  float im;
} x, y;
```

```
x.re =10;
x.im=-10;
y.re =2.5;
y.im= - 0.3;
```

```
struct frutas
{
  char nome[10];
Int calorias;
} ;
```

```
struct frutas laranja, uva;

laranja.nome="laranja";
uva.nome="uva";
laranja.calorias = 70;
uva.calorias = 79;
```

Declarando structs em linguagem C

- O “.” é o operador de acesso a membros da estrutura.

- O uso de typedef é bastante comum na definição dos tipos de estrutura.
  - typedef struct frutas frutas;

    frutas laranja, uva;
  - typedef struct complex compl;

    compl x, y;

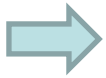
- O nome usado para identificar a estrutura (“etiqueta”) é separado de outros identificadores. Por isso, podemos repetir frutas como um identificador para o typedef anterior.

Declarando structs em linguagem C

- Os nomes dentro da estrutura devem ser únicos, mas podem ser repetidos em estruturas diferentes

```
struct frutas
{
  char *nome;
  int calorias;
};
```

```
struct vegetais;
{
  char *nome;
  int calorias;
};
```



```
struct frutas uva;
struct vegetais  alface;
uva.nome="uva";
uva.calorias = 79;
uva.nome="alface";
uva.calorias = 4;
```

- Estruturas podem conter vetores ou mesmo outras estruturas entre seus membros.

```
struct frutas
{
  char *nome;
  int calorias;
} salada_fruta[20];
```

Declarando structs em linguagem C

- A “etiqueta” da estrutura é opcional, mas uma estrutura declarada sem ela não poderá ser utilizada em outras declarações.

```
struct {
  int dia, mes, ano;
  char nome_dia[4];
  char nome_mes[4];
} ontem, hoje, amanha;
```

- Apenas as variáveis ontem, hoje e amanha possuem a estrutura acima.

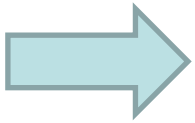
Declarando structs em linguagem C

- O uso da “etiqueta” sem a declaração das variáveis estabelece um *template*.

```
struct  data{
  int dia, mes, ano;
  char nome_dia[4];
  char nome_mes[4];
};
struct data ontem, hoje, amanha;
```

- Quando typedef é utilizado, a “etiqueta” da estrutura pode deixar de ser importante.

```
typedef struct {
  float re;
 float im;
} complex;
```



```
complex a, b, c[100];
```

```
#include <stdio.h>
typedef struct{
    char *nome;
    int calorias;
    char *tipo[2];
}frutas;

int main(void){
frutas laranja, uva;

laranja.nome = "laranja";
laranja.tipo[0]="lima";
laranja.tipo[1]="pera";
laranja.calorias=70;
uva.nome = "uva";
uva.calorias=79;
uva.tipo[0]="italia";
uva.tipo[1]="rubi";
```

```
laranja = uva;
printf("\nfruta=%s calorias=%d tipos= %s
    %s", laranja.nome, laranja.calorias,
    laranja.tipo[0], laranja.tipo[1]);
return 0;
}
```

Saída:
fruta=uva calorias=79 tipos= italia   rubi

- A cada membro de laranja foi atribuído o valor correspondente do respectivo membro de uva.

Acessando membros da estrutura

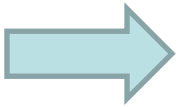
Arquivo class_info.h

```
#define CLASS_SIZE 100
struct aluno{
  char *nome;
  int   id;
  char grade;
};
```

Arquivo main.c

```
#include “class_info.h”
int main(void)
{
  struct aluno tmp,
         class[CLASS_SIZE];
   ....
}
```

- Valores podem ser atribuidos a variável tmp fazendo:



```
tmp.grade=‘A’;
tmp.nome=“Casanova”;
tmp.id=910017;
```

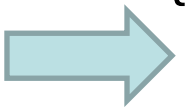
Acessando membros da estrutura

Arquivo class_info.h

```
#define CLASS_SIZE 100
struct aluno{
  char *nome;
  int   id;
  char grade;
};
```

Arquivo main.c

```
#include “class_info.h”
int main(void)
{
  struct aluno tmp,
          class[CLASS_SIZE];
   ....
}
```



- Podemos ter uma função fail que conta o total de alunos reprovados.

Arquivo main.c

```
#include “class_info.h”
int fail(struct aluno class[ ])
{
  int i, cnt = 0;
  for(i=0; i<CLASS_SIZE; ++i)
    cnt += class[ i ].grade==‘F’;
  return cnt;
}
```

Acessando membros da estrutura

- O operador -> permite o acesso à estrutura via ponteiro.
- Se um ponteiro é atribuído ao endereço da estrutura, cada membro da estrutura pode ser acessado fazendo:

ptr->membro_da_estrutura; $\Leftrightarrow$ *(ptr).membro_da_estrutura;

- Apenas estrutura pode ser usada com o operador “.”, um ponteiro para uma estrutura não pode. Logo:

*(ptr).membro_da_estrutura; $\neq$ *(ptr.membro_da_estrutura);

Acessando membros da estrutura

- Exemplo1:

```
struct complex{
 double re;
 double im;
}
...
typedef struct complex cplx;
...
void add(cplx  *a, cplx  *b,  cplx *c)
{
 a-> re = b->re   +  c->re;
 b-> im = b->im  +  c->im;
}
```

Exemplo2:

```
struct student tmp, *p = &tmp;
tmp.grade = 'A';
tmp.nome = "Casanova";
tmp.id=910017;
```

Expressão	Equivalente	Valor
tmp.grade	p->grade	A
tmp.nome	tmp->nome	Casanova
(*p).id	p->id	910017
*p->nome+1	(*(p->nome))+1	D
*(p->nome+2	(p->nome[2])	s

Estruturas com funções

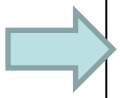
- Estruturas podem ser repassadas como argumentos de funções e podem ser retornadas por uma função.
- Quando repassada como argumento, uma cópia da estrutura é manipulada pela função.
- Ainda que um dos membros da estrutura seja um vetor ou matriz, uma cópia dessas estruturas é realizada.
- Logo, uma estrutura com muitos membros, onde vários deles são vetores com muitas entradas, a passagem da estrutura como argumento pode se tornar ineficiente.
- Uma alternativa é passar o endereço da estrutura como argumento.

Estruturas com funções

•Exemplo: Suponha uma aplicação em alguma grande empresa.

```
struct dept{
    char            dept_nome[25];
    int             dept_no;
}

 typedef struct{
    char            nome[25];
    int             id_funcionario;
    struct  dept    departamento;
    struct  ender   *a_ptr;
    .....
} dados_func;
```



```
dados_func  updat(dada_func   e)
{ ......
   printf("Numero do depto:");
   scanf("%d", &n);
   e.departamento.dept_no = n;
   ....
   return e;
}
```

Estruturas com funções

- O membro departamento na estrutura é uma estrutura. O compilador precisa saber o tamanho de cada membro.

- Logo, a declaração de struct dept precisa ocorrer antes.

- O membro *a_ptr é um ponteiro para uma estrutura. Como o compilador já conhece o tamanho do ponteiro, essa estrutura não precisa ser definida antes.

Estruturas com funções

- Podemos acessar um membro da estrutura dentro de outra estrutura fazendo:

  e.departamento.depto_no ⇔ (e.departamento).dept_no

- Abaixo temos um possível uso de update na função main()

```
dados_func   x;
x = update(x);
```

- Uma cópia da variável **x** é repassada para a função que retorna uma estrutura **e**.
- A estrutura **e** é copiada membro-a-membro para **x**.

Estruturas com funções

- Uma alternativa seria a passagem por referência:

```
void  update(dada_func   *p)
{ ......
   printf("Numero do depto:");
   scanf("%d", &n);
   p->departamento.dept_no = n;
   ....
}
```

- p->departamento.dept_no $\Leftrightarrow$ (p->departamento).dept_no

- Na função main() uma possível chamada seria:

```
dados_func  x;
update(&x);
```

Inicializando estruturas

- Todas as variáveis externas e estáticas, não iniciadas pelo programa, são iniciadas automaticamente pelo sistema com valor zero.
- Isso inclui as estruturas.
- Uma estrutura pode ser iniciada pelo usuário da mesma forma que vetores e matrizes.
- Todos os membros da estrutura não inicializados ficam com valor zero. No caso de ponteiros, recebem valor NULL.

Inicializando estruturas

- Exemplos:

```
Complex a[3][3]={ { {1.0, -0.1}, {2.0,0.2},{3.0,0.3}},
                  {{4.0,-0.4},{5.0,0.5},{6.0,0.6}} ,
                 }; /*a[2][ ] é iniciado com valor zero */

struct  fruta frt={“ameixa”, 150};

struct  ender{
   char  *rua;
   char *cidade_UF;
   long  cep
} endereco = {“Trabalhador são-carlense, 400”, “São Carlos”, 13566-590 };

struct ender   ender_anterior = {0};
```

Exemplo de Pilha

```
#include <stdio.h>

#define MAX_LEN 1000
#define EMPTY   -1
#define FULL    (MAX_LEN -1)

typedef enum boolean {false,
    true}  boolean;

typedef struct stack{
   char s[MAX_LEN];
   int top;
} stack;
```

```
void reset(stack *stk)
{
  stk -> top = EMPTY;
}
void push(char c, stack *stk)
{
   stk->top++;
   stk-> s[stk->top]=c;
}
char pop(stack *stk)
{
   return (stk->s[stk->top--]);
}
```

```
char top(const stack *stk)
{
    return (stk-> s[stk->top]);
}
boolean empty(const stack *stk)
{
    return ((boolean) (stk->top ==
     EMPTY));
}
boolean full (const stack *stk)
{
    return ((boolean) (stk->top
     ==FULL));
}
```



```
int main(void){
 char str[] = "Meu nome eh Joao";
 int i;
 stack s;

 reset (&s);
 printf("String: %s\n", str);
 for(i=0; str[i]!='\0'; ++i)
    if(!full(&s))
     push(str[i],&s);

 printf("Pilha:");
 while(!empty(&s))
  putchar(pop(&s));
 putchar('\n');

 return 0;
}
```

Unions

- Uma união define um conjunto de valores que podem ser armazenados em uma mesma área de memória.
- O programador é responsável por interpretar os valores armazenados corretamente.
- Logo, estruturas do tipo union permitem armazenar objetos de diferentes tipos e tamanhos em um mesmo pedaço de memória.
- Exemplo:

```
union int_or_float{
  int i;
  float f;
};
```

Unions

- Exemplo:

```
typedef union int_or_float{
 int i;
 float f;
}number;
int main(void)
{
 number n;
 n.i = 4444;
 printf("i: %10d     f:%16.10e\n", n.i, n.f);
 n.f = 4440.0;
 printf("i: %10d     f:%16.10e\n", n.i, n.f);
return 0;
}
```

```
i:          4444    f: 6.227370375e-41

i: 1166729216    f: 4.4440000000e+03
```

Unions

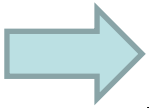
- Os membros de unions podem ser estruturas e outras unions.
- Exemplo:

```
struct  flor{
 char     *nome;
 enum {verm, branca, azul}  cor;
};
```

```
struct  frutas{
 char     *nome;
 int * calorias;
};
```

```
struct  vegetais{
 char     *nome;
 int      calorias;
 int      tempo_preparo;
};
```

```
union
flores_frutas_ou_vegetais {
 struct   flr;
 struct   frt;
 struct   veg;
};
```



```
union flores_frutas_ou_vegetais  ffv;

ffv.veg.tempo_preparo = 7;
```

Referências

Ascencio AFG, Campos EAV. Fundamentos de programação de computadores. São Paulo : Pearson Prentice Hall, 2006. 385 p.

Kelley, A.; Pohl, I*., A Book on C*: programming in C. 4ª Edição. Massachusetts: Pearson, 2010, 726p.

Kernighan, B.W.; Ritchie, D.M. C, *A Linguagem de Programação*: padrão ANSI. 2ª Edição. Rio de Janeiro: Campus, 1989, 290p.

Schildt, Herbet, *C Completo e Total*, Pearson, 2006,

FIM Aula 18