Elementos de programação em C

Estruturas e uniões

Francisco A. C. Pinheiro, *Elementos de Programação em C*, Bookman, 2012.

Visite os sítios do livro para obter material adicional: www.bookman.com.br e www.facp.pro.br/livroc

Sumário

Declaração de estruturas

As estruturas são declaradas através de variáveis do tipo estrutura que especifica os seus componentes.

Declaração de estruturas

As estruturas são declaradas através de variáveis do tipo estrutura que especifica os seus componentes.

Exemplo

Para o tipo estrutura ao lado as seguintes declarações são possíveis:

```
struct {
  int matr;
  float nota1;
  float nota2;
}
```

```
struct {
  int matr;
  float nota1;
  float nota2;
} aluno,
  aluno_regular;
```

```
struct r_aluno {
  int matr;
  float nota1;
  float nota2;
} aluno;
struct r_aluno
       aluno_regular,
       aluno_especial;
```

```
typedef struct {
  int matr;
  float nota1;
  float nota2;
} tp_r_aluno;
tp_r_aluno aluno,
   aluno_regular;
```

Operador de seleção direta

⟨*OpSelecaoDireta*⟩ ::= ⟨*IdEstrutura*⟩ **.** ⟨*IdComponente*⟩

⟨*IdEstrutura*⟩ ::= Identificador da estrutura, geralmente uma variável de tipo estrutura.

⟨*IdComponente*⟩ ::= Identificador do componente.

Operador de seleção direta

⟨*OpSelecaoDireta*⟩ ::= ⟨*IdEstrutura*⟩ . ⟨*IdComponente*⟩

⟨*IdEstrutura*⟩ ::= Identificador da estrutura, geralmente uma variável de tipo estrutura.

⟨*IdComponente*⟩ ::= Identificador do componente.

Exemplo

`aluno.nota1`	Componente `nota1` da estrutura armazenada em `aluno`.
`aluno.matr`	Componente `matr` da mesma estrutura.

Operador de seleção indireta

⟨*OpSelecaoIndireta*⟩ ::= ⟨*PtrEstrutura*⟩ -> ⟨*IdComponente*⟩

⟨*PtrEstrutura*⟩ ::= Variável do tipo ponteiro para estrutura.

⟨*IdComponente*⟩ ::= Identificador do componente.

Operador de seleção indireta

⟨*OpSelecaoIndireta*⟩ ::= ⟨*PtrEstrutura*⟩ -> ⟨*IdComponente*⟩

⟨*PtrEstrutura*⟩ ::= Variável do tipo ponteiro para estrutura.

⟨*IdComponente*⟩ ::= Identificador do componente.

Exemplo

`ptr_aluno->nota1`	Componente `nota1` da estrutura apontada por `ptr_aluno`.
`ptr_aluno->matr`	Componente `matr` da mesma estrutura.

Usando estruturas como valor de retorno e argumento

Exemplo

O programa ao lado lê os dados de um aluno, armazenando-os em uma estrutura e imprimindo-os em seguida.

```
#include <stdio.h>
struct r_aluno {
  int matr;
  float nota1;
  float nota2;
};
struct r_aluno ler_aluno(void);
void imp_aluno(struct r_aluno);
int main(void) {
  struct r_aluno aluno = ler_aluno();
  imp_aluno(aluno);
  return 0;
}
```

continua...

Usando estruturas como valor de retorno e argumento

Exemplo

...continuação

```
struct r_aluno ler_aluno(void) {
  struct r_aluno al;
  printf("Digite os dados do aluno\n");
  printf("Matricula: ");
  scanf("%d", &al.matr);
  printf("Primeira nota: ");
  scanf("%f", &al.nota1);
  printf("Segunda nota: ");
  scanf("%f", &al.nota2);
  return al;
}
void imp_aluno(struct r_aluno al) {
  printf("Matr: %d Notas: %5.2f %5.2f ",
          al.matr, al.nota1, al.nota2);
  printf("Media: %5.2f\n",
            (al.nota1 + al.nota2) / 2);
}
```

Ponteiros para estruturas

Exemplo

O programa ao lado modifica o exemplo anterior, usando um ponteiro para a estrutura aluno.

```
#include <stdio.h>
struct r_aluno {
  int matr;
  float nota1;
  float nota2;
};
void ler_aluno(struct r_aluno *);
void imp_aluno(struct r_aluno *);
int main(void) {
  struct r_aluno aluno;
  ler_aluno(&aluno);
  imp_aluno(&aluno);
  return 0;
}
```

continua...

Ponteiros para estruturas

Exemplo

...continuação

```
void ler_aluno(struct r_aluno *al) {
  printf("Digite os dados do aluno\n");
  printf("Matricula: ");
  scanf("%d", &al->matr);
  printf("Primeira nota: ");
  scanf("%f", &al->nota1);
  printf("Segunda nota: ");
  scanf("%f", &al->nota2);
}
void imp_aluno(struct r_aluno *al) {
  printf("Matr: %d Notas: %5.2f %5.2f ",
         al->matr, al->nota1, al->nota2);
  printf("Media: %5.2f\n",
            (al->nota1 + al->nota2) / 2);
}
```

Ponteiros para estruturas

- O conteúdo do endereço apontado por um ponteiro para estrutura é a própria estrutura.
- Se `al` é um ponteiro para uma estrutura, então `*al` é a própria estrutura apontada por `al`.

Ponteiros para estruturas

Exemplo

Considerando que `al` é um ponteiro para a estrutura `aluno` dos exemplos anteriores, o que representam as expressões a seguir?

```
&(*al)

al->nota1
(*al).nota1
(&(*al))->nota1

&(*al).nota1
&((*al).nota1).
*al.nota1
```

Ponteiros para estruturas

Exemplo

Considerando que `al` é um ponteiro para a estrutura `aluno` dos exemplos anteriores, o que representam as expressões a seguir?

`&(*al)`	Endereço da estrutura armazenada no endereço apontado por `al`. Isto é, `&(*al)` = `al`.
`al->nota1`	
`(*al).nota1`	
`(&(*al))->nota1`	
`&(*al).nota1`	
`&((*al).nota1)`	
`*al.nota1`	

Ponteiros para estruturas

Exemplo

Considerando que `al` é um ponteiro para a estrutura `aluno` dos exemplos anteriores, o que representam as expressões a seguir?

`&(*al)`	Endereço da estrutura armazenada no endereço apontado por `al`. Isto é, `&(*al)` = `al`.
`al->nota1`	Componente `nota1` da estrutura apontada por `al`.
`(*al).nota1`	
`(&(*al))->nota1`	
`&(*al).nota1`	
`&((*al).nota1)`	
`*al.nota1`	

Ponteiros para estruturas

Exemplo

Considerando que `al` é um ponteiro para a estrutura `aluno` dos exemplos anteriores, o que representam as expressões a seguir?

`&(*al)`	Endereço da estrutura armazenada no endereço apontado por `al`. Isto é, `&(*al)` = `al`.
`al->nota1`	Componente `nota1` da estrutura apontada por `al`.
`(*al).nota1`	Componente `nota1` da estrutura `*al`.
`(&(*al))->nota1`	

`&(*al).nota1`

`&((*al).nota1)`.

`*al.nota1`

Ponteiros para estruturas

Exemplo

Considerando que `al` é um ponteiro para a estrutura `aluno` dos exemplos anteriores, o que representam as expressões a seguir?

`&(*al)`	Endereço da estrutura armazenada no endereço apontado por `al`. Isto é, `&(*al)` = `al`.
`al->nota1`	Componente `nota1` da estrutura apontada por `al`.
`(*al).nota1`	Componente `nota1` da estrutura `*al`.
`(&(*al))->nota1`	Componente `nota1` da estrutura cujo endereço é `&(*al)`.
`&(*al).nota1`	
`&((*al).nota1)`	
`*al.nota1`	

Ponteiros para estruturas

Exemplo

Considerando que `al` é um ponteiro para a estrutura `aluno` dos exemplos anteriores, o que representam as expressões a seguir?

`&(*al)`	Endereço da estrutura armazenada no endereço apontado por `al`. Isto é, `&(*al)` = `al`.
`al->nota1`	Componente `nota1` da estrutura apontada por `al`.
`(*al).nota1`	Componente `nota1` da estrutura `*al`.
`(&(*al))->nota1`	Componente `nota1` da estrutura cujo endereço é `&(*al)`.
`&(*al).nota1`	Endereço do componente `nota1` da estrutura `*al`.
`&((*al).nota1)`	
`*al.nota1`	

Ponteiros para estruturas

Exemplo

Considerando que `al` é um ponteiro para a estrutura `aluno` dos exemplos anteriores, o que representam as expressões a seguir?

`&(*al)`	Endereço da estrutura armazenada no endereço apontado por `al`. Isto é, `&(*al)` = `al`.
`al->nota1`	Componente `nota1` da estrutura apontada por `al`.
`(*al).nota1`	Componente `nota1` da estrutura `*al`.
`(&(*al))->nota1`	Componente `nota1` da estrutura cujo endereço é `&(*al)`.
`&(*al).nota1`	Endereço do componente `nota1` da estrutura `*al`.
`&((*al).nota1)`.	Endereço do componente `nota1` da estrutura `*al`.
`*al.nota1`	

Ponteiros para estruturas

Exemplo

Considerando que `al` é um ponteiro para a estrutura `aluno` dos exemplos anteriores, o que representam as expressões a seguir?

`&(*al)`	Endereço da estrutura armazenada no endereço apontado por `al`. Isto é, `&(*al)` = `al`.
`al->nota1`	Componente `nota1` da estrutura apontada por `al`.
`(*al).nota1`	Componente `nota1` da estrutura `*al`.
`(&(*al))->nota1`	Componente `nota1` da estrutura cujo endereço é `&(*al)`.
`&(*al).nota1`	Endereço do componente `nota1` da estrutura `*al`.
`&((*al).nota1)`.	Endereço do componente `nota1` da estrutura `*al`.
`*al.nota1`	Expressão inválida.

Estruturas com componente flexível

- O último componente de uma estrutura contendo mais de um componente nomeado pode ser de um tipo vetor incompleto.
  - *componente (vetor) flexível*.
- Antes do uso deve haver alocação explícita de espaço em memória para armazenar os elementos do vetor.
- O tamanho de um tipo estrutura com componente flexível é o mesmo que se o componente não existisse.

Estruturas com componente flexível

Exemplo

No programa ao lado a estrutura apontada pela variável `reg_a` é alocada com 4 elementos para o vetor `vendas_mes`.

```
#include <stdio.h>
#include <stdlib.h>
struct r1 {
  int matr;
  int vendas_mes[];
};
int main(void) {
  struct r1 *reg_a;
  reg_a = (struct r1 *)malloc(
     sizeof(struct r1) + sizeof(int[4]));
  for (int i = 0; i < 4; i++) {
    printf("%d\n", reg_a->vendas_mes[i]);
  }
  return 0;
}
```

Vetores de estruturas

Um vetor de estruturas é um vetor cujos elementos são de um tipo estrutura.

Exemplo

```
struct {
  int matr;
  float nota1;
  float nota2;
} v_alunos[20];
```

```
typedef struct {
  int matr;
  float nota1;
  float nota2;
} tp_r_aluno;
tp_r_aluno v_alunos[20];
```

Vetores de estruturas

Exemplo. O programa ao lado usa um vetor de estruturas para armazenar (e imprimir) os dados de dez alunos.

```
#include <stdio.h>
#define QTD_AL (10)
struct r_aluno {
  int matr;
  float nota1;
  float nota2;
};
void imp_alunos(struct r_aluno []);
int main(void) {
  struct r_aluno v_alunos[QTD_AL];
  printf("Digite os dados dos %d alunos\n",
         QTD_AL);
  for (int i = 0; i < QTD_AL; i++) {
    printf("Matricula %d: ", (i + 1));
    scanf("%d", &v_alunos[i].matr);
    printf("Primeira nota: ");
    scanf("%f", &v_alunos[i].nota1);
    printf("Segunda nota: ");
    scanf("%f", &v_alunos[i].nota2);
  }
```

continua...

Vetores de estruturas

Exemplo.

continuação...

```
  imp_alunos(v_alunos);
  return 0;
}
void imp_alunos(struct r_aluno al[]) {
  for (int i = 0; i < QTD_AL; i++) {
    printf("Matr: %d Notas: %5.2f %5.2f",
     al[i].matr, al[i].nota1, al[i].nota2);
    printf(" Media: %5.2f\n",
          (al[i].nota1 + al[i].nota2) / 2);
  }
}
```

Vetores de estruturas

Exemplo

Para o vetor `v_alunos` ao lado, o que representam as seguintes expressões?

```
struct r_aluno {
  int matr;
  float nota1, nota2;
} v_alunos[1000];
```

`v_alunos`

`*(v_alunos + 2)`

`&v_alunos[2]`

`v_alunos[2].matr`

`(*v_alunos).matr`

Vetores de estruturas

Exemplo

Para o vetor `v_alunos` ao lado, o que representam as seguintes expressões?

```
struct r_aluno {
  int matr;
  float nota1, nota2;
} v_alunos[1000];
```

`v_alunos` — Ponteiro para a estrutura, apontando inicialmente para o primeiro elemento.

`*(v_alunos + 2)`

`&v_alunos[2]`

`v_alunos[2].matr`

`(*v_alunos).matr`

Vetores de estruturas

Exemplo

Para o vetor `v_alunos` ao lado, o que representam as seguintes expressões?

```
struct r_aluno {
  int matr;
  float nota1, nota2;
} v_alunos[1000];
```

`v_alunos`	Ponteiro para a estrutura, apontando inicialmente para o primeiro elemento.
`*(v_alunos + 2)`	Estrutura armazenada no elemento apontado por `(v_alunos + 2)`.
`&v_alunos[2]`	
`v_alunos[2].matr`	
`(*v_alunos).matr`	

Vetores de estruturas

Exemplo

Para o vetor `v_alunos` ao lado, o que representam as seguintes expressões?

```
struct r_aluno {
  int matr;
  float nota1, nota2;
} v_alunos[1000];
```

`v_alunos`	Ponteiro para a estrutura, apontando inicialmente para o primeiro elemento.
`*(v_alunos + 2)`	Estrutura armazenada no elemento apontado por `(v_alunos + 2)`.
`&v_alunos[2]`	Endereço do terceiro elemento do vetor.
`v_alunos[2].matr`	
`(*v_alunos).matr`	

Vetores de estruturas

Exemplo

Para o vetor `v_alunos` ao lado, o que representam as seguintes expressões?

```
struct r_aluno {
  int matr;
  float nota1, nota2;
} v_alunos[1000];
```

`v_alunos`	Ponteiro para a estrutura, apontando inicialmente para o primeiro elemento.
`*(v_alunos + 2)`	Estrutura armazenada no elemento apontado por `(v_alunos + 2)`.
`&v_alunos[2]`	Endereço do terceiro elemento do vetor.
`v_alunos[2].matr`	Componente `matr` da estrutura armazenada no terceiro elemento do vetor.
`(*v_alunos).matr`	

Vetores de estruturas

Exemplo

Para o vetor `v_alunos` ao lado, o que representam as seguintes expressões?

```
struct r_aluno {
  int matr;
  float nota1, nota2;
} v_alunos[1000];
```

`v_alunos`	Ponteiro para a estrutura, apontando inicialmente para o primeiro elemento.
`*(v_alunos + 2)`	Estrutura armazenada no elemento apontado por `(v_alunos + 2)`.
`&v_alunos[2]`	Endereço do terceiro elemento do vetor.
`v_alunos[2].matr`	Componente `matr` da estrutura armazenada no terceiro elemento do vetor.
`(*v_alunos).matr`	Componente `matr` da estrutura armazenada no elemento apontado por `v_alunos`.

Vetores de estruturas

Exemplo

Se pa aponta para o segundo elemento do vetor ao lado, e sabendo que os operadores -> e . têm maior precedência que ++, o que representam as seguintes expressões?

```
pa++->matr

++pa->matr

++(pa->matr)

(++pa)->matr
```

```
struct r_aluno {
  int matr;
  float nota1, nota2;
} v_alunos[1000];
```

Vetores de estruturas

Exemplo

Se `pa` aponta para o segundo elemento do vetor ao lado, e sabendo que os operadores `->` e `.` têm maior precedência que `++`, o que representam as seguintes expressões?

```
struct r_aluno {
  int matr;
  float nota1, nota2;
} v_alunos[1000];
```

`pa++->matr` — Componente `matr` do segundo elemento. Ponteiro é incrementado.

`++pa->matr`

`++(pa->matr)`

`(++pa)->matr`

Vetores de estruturas

Exemplo

Se `pa` aponta para o segundo elemento do vetor ao lado, e sabendo que os operadores `->` e `.` têm maior precedência que `++`, o que representam as seguintes expressões?

```
struct r_aluno {
  int matr;
  float nota1, nota2;
} v_alunos[1000];
```

`pa++->matr`	Componente `matr` do segundo elemento. Ponteiro é incrementado.
`++pa->matr`	Componente `matr` do segundo elemento. Componente é incrementado.
`++(pa->matr)`	
`(++pa)->matr`	

Vetores de estruturas

Exemplo

Se `pa` aponta para o segundo elemento do vetor ao lado, e sabendo que os operadores `->` e `.` têm maior precedência que `++`, o que representam as seguintes expressões?

```
struct r_aluno {
  int matr;
  float nota1, nota2;
} v_alunos[1000];
```

`pa++->matr`	Componente `matr` do segundo elemento. Ponteiro é incrementado.
`++pa->matr`	Componente `matr` do segundo elemento. Componente é incrementado.
`++(pa->matr)`	Componente `matr` do segundo elemento. Componente é incrementado.
`(++pa)->matr`	

Vetores de estruturas

Exemplo

Se `pa` aponta para o segundo elemento do vetor ao lado, e sabendo que os operadores `->` e `.` têm maior precedência que `++`, o que representam as seguintes expressões?

```
struct r_aluno {
  int matr;
  float nota1, nota2;
} v_alunos[1000];
```

`pa++->matr`	Componente `matr` do segundo elemento. Ponteiro é incrementado.
`++pa->matr`	Componente `matr` do segundo elemento. Componente é incrementado.
`++(pa->matr)`	Componente `matr` do segundo elemento. Componente é incrementado.
`(++pa)->matr`	Componente `matr` do terceiro elemento. Ponteiro é incrementado.

Iniciando estruturas

- As variáveis do tipo estrutura podem ser iniciadas com uma relação de iniciação.

Exemplo

```
struct {
  int matr;
  float nota1;
  float nota2;
} aluno = {130518, 4.57, 8.33, 9.54},
  aluno_regular = {234, 7.65};
```

Iniciação seletiva e componentes agregados

- A iniciação seletiva se faz indicando o componente com a notação $.\langle nome\_componente \rangle$
- Os componentes agregados podem ser especificados com chaves.

Exemplo

```
struct {
  int matr;
  struct {
    char sexo;
    int  rg;
  } sit;
  float nota1;
  float nota2;
} reg_a = {1111, 'm', 715, 8.54, 5.73},
  reg_b = {2222, 'f'},
  reg_c = {.sit = {'f', 32}, 215.3},
  reg_d = {3333, {'m', 234}, 86.44};
```

Iniciação com literais compostos

- As variáveis automáticas podem ser iniciadas com literais compostos.

Exemplo

```
struct r_aula {
  char topico[20];
  int ini;
  int fim;
} aula =
   (struct r_aula){"Programacao", 10, 12};
```

Declaração de uniões

As uniões são declaradas por meio do tipo união que especifica os seus componentes.

Declaração de uniões

As uniões são declaradas por meio do tipo união que especifica os seus componentes.

Exemplo

Para o tipo união ao lado as seguintes declarações são possíveis:

```
union {
  unsigned int regra;
  char norma;
}
```

```
union {
  unsigned int
           regra;
  char norma;
} id;
```

```
union doc_id {
  unsigned int
           regra;
  char norma;
};
union doc_id id;
```

```
typedef union {
  unsigned int
           regra;
  char norma;
} tp_doc_id;
tp_doc_id id;
```

Referenciando os componentes de uma união

Os componentes de uma união são referidos do mesmo modo que os componentes de uma estrutura:

- Por meio do operador de seleção direta
- Por meio do operador de seleção indireta

Usando os componentes apropriados

- O valor de uma união deve ser acessado através do componente utilizado para armazená-lo.
- O comportamento é indefinido se o valor de uma união é acessado com um componente diferente do componente usado para armazená-lo.

Usando os componentes apropriados

Exemplo

Para a união ao lado, após a atribuição `tst.num = 23`, as expressões a seguir têm o significado descrito:

```
union u {
  int num;
  char texto[20];
} tst;
```

`printf("%d", tst.num)` — Correto: imprime o conteúdo da variável `tst` como um valor do tipo `int`.

`printf("%s", tst.texto)` — Errado: imprime o conteúdo da variável `tst` como um valor do tipo `char [20]`.

`printf("%d", tst)` — Inapropriado: imprime o conteúdo da variável `tst`, convertendo-o do tipo `union u` em um valor do tipo `int`.

Usando os componentes apropriados

Exemplo

Em cada impressão do programa ao lado apenas um valor está corretamente referenciado.

```
#include <stdio.h>
union r {
  float a;
  double b;
  int c;
} tst;
int main(void) {
  tst.a = 34.5;
  printf("%g %g %d\n",tst.a, tst.b, tst.c);
  tst.b = 34.5;
  printf("%g %g %d\n",tst.a, tst.b, tst.c);
  tst.c = 34.5;
  printf("%g %g %d\n",tst.a, tst.b, tst.c);
  return 0;
}
```

O programa produz a seguinte saída:

```
34.5 5.47401e-315 1107951616
0 34.5 0
4.76441e-44 34.5 34
```

Uniões como componentes de estruturas

É comum usar uniões como componentes de estruturas, juntamente com um outro componente indicando como a união deve ser interpretada.

Exemplo

```
union doc_id {
  unsigned int
           regra;
  char     norma;
};
struct r_doc {
  char tipo;
  union doc_id id;
  char *resumo;
};
```

```
struct r_doc {
  char tipo;
  union {
    unsigned int
             regra;
    char     norma;
  } id;
  char *resumo;
};
```

```
typedef union {
  unsigned int
           regra;
  char     norma;
} tp_id;
struct r_doc {
  char tipo;
  tp_id id;
  char *resumo;
};
```

Uniões como componentes de estruturas

Exemplo. O programa ao lado lê o tipo e a identificação de um documento. Existem dois formatos para a identificação, variando de acordo com o tipo. Ambos são armazenados em uma mesma variável do tipo união.

continua...

```
#include <stdio.h>
typedef union {
  unsigned int regra;
  char norma;
} tp_id;
struct r_doc {
  char tipo;
  tp_id id;
  char *resumo;
};
void obtem_id(tp_id *, char);
void imp_doc(struct r_doc *);
void limpa_linha(void);
int main(void) {
  struct r_doc doc;
  printf("Digite o tipo de documento: ");
  doc.tipo = getchar();
  limpa_linha();
  obtem_id(&doc.id, doc.tipo);
  imp_doc(&doc);
  return 0;
}
```

Uniões como componentes de estruturas

...continuação,

```
void obtem_id(tp_id *id, char tipo) {
  if (tipo == 'r') {
    printf("id regra (numerico): ");
    scanf("%u", &id->regra);
  } else {
    if (tipo == 'n') {
      printf("id norma (char): ");
      scanf("%c", &id->norma);
    }
  }
}
```

continua...

Uniões como componentes de estruturas

...continuação.

```
void imp_doc(struct r_doc *d) {
  printf("tipo: %c id: ", d->tipo);
  if (d->tipo == 'r') {
    printf("%u\n", d->id.regra);
  } else {
    if (d->tipo == 'n') {
      printf("%c\n", d->id.norma);
    } else {
      printf("tipo invalido\n");
    }
  }
}
void limpa_linha() {
  while (getchar() != '\n') { };
}
```

Iniciando uniões

As uniões são explicitamente iniciadas com o valor correspondente ao seu primeiro componente especificado entre chaves, podendo haver a iniciação seletiva, indicando-se um componente específico da união.

Iniciando uniões

As uniões são explicitamente iniciadas com o valor correspondente ao seu primeiro componente especificado entre chaves, podendo haver a iniciação seletiva, indicando-se um componente específico da união.

Exemplo

Para a declaração de tipo ao lado,

```
union doc_id {
  unsigned int regra;
  char norma;
};
```

Iniciação	Corresponde a
`union doc_id id = {1234};`	`id.regra = 1234;`
`union doc_id id = {'t'};`	`id.regra = 't';`
`union doc_id id = {.norma = 't'};`	`id.norma = 't';`

Campos de bits

- Define componentes de tipo inteiro com uma quantidade limitada de bits.
- O tipo de um campo de bits deve ser
  - `_Bool`,
  - `signed int`,
  - `unsigned int`, ou
  - algum outro tipo dependente da implementação.
- O tipo `int` será implementado como `signed int` ou `unsigned int`, dependendo da implementação.

Campos de bits

- Define componentes de tipo inteiro com uma quantidade limitada de bits.
- O tipo de um campo de bits deve ser
  - `_Bool`,
  - `signed int`,
  - `unsigned int`, ou
  - algum outro tipo dependente da implementação.
- O tipo `int` será implementado como `signed int` ou `unsigned int`, dependendo da implementação.

Exemplo

A estrutura ao lado usa dois campos de bits.

```
struct r_arq {
  char id[9];
  _Bool grv: 1;
  unsigned int perm: 3;
}
```

Campos de bits

São altamente dependentes da implementação:

- Não podem ser acessados com o operador de endereço (&).
- Podem ser não-nomeados.
- Os campos não-nomeados podem ser especificados com zero bits.
  - Um campo de bits pode ser alocado na mesma unidade de armazenamento que o campo precedente, ou não.
  - O uso de um campo com 0 bits faz com que o próximo não seja alocado na mesma unidade que o campo precedente.

Compatibilidade de estruturas e uniões

Duas estruturas ou uniões (declaradas em diferentes unidades de compilação) são compatíveis se

1. possuem a mesma etiqueta.
2. Adicionalmente, se os tipos são completos, deve existir uma correspondência entre os seus componentes tal que os componentes correspondentes devem
   - ser declarados na mesma ordem (apenas para estruturas);
   - ter tipos compatíveis;
   - ter nomes iguais (se forem nomeados); e
   - ser do mesmo tamanho (se forem campos de bits).

Compatibilidade de estruturas e uniões

Exemplo

A compilação do seguinte programa gera um executável com comportamento indefinido.

prgA.c

```
#include <stdio.h>
void fun (void);
struct s1 {
int a;
int b;
};
struct s1
   var1 = {33, 44};
struct s1 var2;
int main(void) {
var2 = var1 ;
fun();
return 0;
}
```

prgB.c

```
#include <stdio.h>
struct s1 {
int a, b;
char c;
};
struct s1 var2 = {111 , 222 , 's'};
void fun(void) {
printf ("%d %d %c\n", var2.a, var2.b,
                                var2.c);
}
```

Bibliografia

ISO/IEC

C Programming Language Standard

ISO/IEC 9899:2011, International Organization for Standardization; International Electrotechnical Commission, 3rd edition, WG14/N1570 Committee final draft, abril de 2011.

Francisco A. C. Pinheiro

Elementos de programação em C

Bookman, Porto Alegre, 2012.

www.bookman.com.br, www.facp.pro.br/livroc