1.Definições de estruturas

As estruturas são tipos derivados de dados, construídas usando objetos de outros tipos.

```
struct carta{
     char *face;

     char *naipe;

};
```

A palavra-chave ***struct*** apresenta a definição da estrutura. O identificador ***carta*** é o tag da estrutura. O tag da estrutura dá o nome da definição da estrutura e é usado com a palavra-chave ***struct*** para declarar as variáveis do tipo da estrutura.

Os membros das estruturas podem ser variáveis dos tipos básicos de dados ou agregadas, como ***arrays*** e outras estruturas. Entretanto, os membros das estruturas podem ser de vários tipos de dado. Por exemplo, uma ***struct*** empregado pode conter membros ***strings*** de caracteres para o primeiro e último nomes, e um membro ***int*** para a idade do empregado.

Uma estrutura não pode conter uma instância de si mesma. Por exemplo, uma variável de tipo ***struct carta*** não pode ser declarada na definição de ***struct carta***. Entretanto, um ponteiro para ***struct*** carta pode ser incluído. Uma estrutura que contém um membro que é ponteiro para o mesmo tipo de estrutura é chamada ***estrutura auto-referencia***. As estruturas auto-referenciadas são usadas para construir vários tipos de estruturas encadeadas de dados.

A definição de estrutura anterior não reserva espaço algum na memória, em vez disso, a definição cria um novo tipo de dado que é usado para declarar variáveis. As variáveis de estruturas são declaradas como variáveis de outros tipos.

As variáveis de um determinado tipo de estrutura também podem ser declaradas colocando uma lista de nomes de variáveis separados por vírgulas, entre a chave final da definição da estrutura e o ponto-e-vírgula que finaliza aquela definição. O nome da tag da estrutura é opcional. Se uma definição de estrutura não possuir um nome de tag e variáveis do tipo da estrutura só podem ser declaradas na sua definição.

As únicas operações válidas que podem ser realizadas em estruturas são: atribuir variáveis de estrutura a variáveis de estruturas do mesmo tipo, obter o endereço (&) de uma variável de estrutura de acesso aos membros de uma variável de estrutura e usar o operador ***sizeof*** para determinar o tamanho de uma variável de estrutura.

As estruturas não podem ser comparadas porque seus membros não são armazenados obrigatoriamente em bytes consecutivos da memória. Algumas vezes há "buracos" em uma estrutura porque os computadores podem armazenar tipos específicos de dados apenas em determinados limites da memória contém limites de meias-palavras, palavras e palavras duplas. Uma palavra é uma unidade-padrão de memória usada para armazenar dados em um computador (normalmente 2 ou 4 bytes)

2.Inicializando estruturas

As estruturas podem ser inicializadas usando listas de inicializadores como ***arrays***. Para inicializar uma estrutura, coloque, depois do nome da variável, um sinal de igual e, entre chaves, uma lista de inicializadores separados por vírgula.

```
struct carta a = {"tres", "copas"};
```

A ***struct*** acima cria a variável ***a*** do tipo ***struct*** carta e inicializa o membro face como ***tres*** e o membro naipe como ***copas***. Se houver menos inicializadores na lista do que os membros na estrutura, os membros restantes são inicializados automaticamente com 0, ou NULL se for um ponteiro. As variáveis de estruturas declaradas foram da definição de uma função são inicializadas com 0 ou NULL se não forem inicializadas explicitamente na declaração externa. As variáveis de estruturas também podem ser inicializadas em instruções de atribuição atribuindo valores a cada um dos membros da estrutura.

3.Acesso a membros de estrutura

São usados dois operadores para acesso a membros de estruturas, o operador de membro de estrutura (.) (também chamado de operador de ponto), e o operador de ponteiro de estrutura (->) (também chamado operador de seta).

O operador de membro de estrutura acessa um membro de uma estrutura por meio do nome da variável da estrutura. Por exemplo, para imprimir o membro "naipe" da estrutura "a" da declaração anterior, use a instrução abaixo.

```
printf ("%s", a.naipe);
```

O operador de ponteiro de estrutura oferece acesso a um membro de uma estrutura por meio de um ponteiro para a estrutura. Assuma que o ponteiro aPtr foi declarado para apontar para "struct" carta e que o endereço da estrutura "a" foi atribuído a aPtr. Para imprimir o membro "naipe" da estrutura "a" com o ponteiro aPtr, use a instrução abaixo.

```
printf("%s", aPtr->naipe);
```

4.Typedef

A palavra-chave ***typedef*** fornece um mecanismo para a criação de sinônimos para tipos de dados definidos previamente. Os nomes dos tipos de estruturas são definidos frequentemente com ***typedef*** para criar nomes mais curtos de tipos.

A instrução abaixo define o novo nome de tipo Carta como sinônimo do tipo ***struct*** carta. Os programadores da linguagem C usam frequentemente ***typedef*** para definir um tipo de estrutura de modo que não é exigido tag de estrutura.

```
typedef struct carta Carta;
```

Por exemplo, a seguinte definição abaixo cria o tipo de estrutura Carta sem a necessidade de uma instrução typedef separada. Agora Carta pode ser usado para declarar variáveis do tipo ***struct*** carta.

```
typedef struct{
     char *face;

     char *naipe;

}Carta;
```

Criar um novo nome com typedef não cria um novo tipo. Typedef simplesmente cria um novo nome de um tipo que pode ser usado como um alias de um nome de um tipo existente. Um nome significativo ajuda a tornar o programa autodocumentado.

Frequentemente, ***typedef*** é usado para criar sinônimos de tipos básicos de dados. Por exemplo, um programa que exija inteiros de 4 bytes pode usar o tipo ***int*** em um sistema e o tipo ***long***. Os programas que devem apresentar portabilidade usam frequentemente ***typedef*** para criar um alias para inteiros de 4 bytes como “Integer”. O alias “Integer” pode ser modificado uma vez no programa para fazer com que ele funcione em ambos os sistemas.

5.Uniões

Uma união é um tipo derivado de dados cujos membros compartilham o mesmo espaço de armazenamento. Para diferentes situações de um programa, algumas variáveis podem não ser apropriadas, mas outras são. Assim sendo, uma união compartilha o espaço em vez de desperdiçar armazenamento em variáveis que não estão sendo usadas.

Os membros de uma união podem ser de qualquer tipo. O número de bytes usados para armazenar uma união deve ser pelo menos o suficiente para conter o maior membro. Na maioria dos casos, as uniões contêm dois ou mais tipos de dados. Apenas um membro, e portanto, apenas um tipo de dado, pode ser referenciado de

cada vez. É responsabilidade do programador assegurar que os dados de uma união sejam referenciados com o tipo apropriado.

Uma união é declarada com a palavra-chave “union” no mesmo formato que uma estrutura. A declaração “union” indica que numero é um tipo de união com membros “int” x e “float” y. Normalmente, a definição da união precede “main” em um programa, portanto, a definição pode ser usada para declarar variáveis em todas as funções do programa.

As operações que podem ser realizadas em uma união são: atribuir uma união a outra união do mesmo tipo, obter o endereço (&) de uma união e ter acesso aos membros de uma união usando o operador de membro de estrutura e o operador de ponteiro de estrutura. As uniões não podem ser comparadas pelas mesmas razões que levam à comparação de estruturas não ser possível.

Por exemplo, com a união precedente, a declaração abaixo é uma inicialização válida da variável de união valor porque a união é inicializada com um “int”, mas a declaração abaixo dessa, seria inválida.

```
union numero valor = {10};

union numero valor = {1.43};
```

6.Campos de bits

A linguagem C fornece a capacidade de especificar o número de bits no qual um membro “unsigned” ou “int” de uma estrutura ou união é armazenado. Os campos de bits permitem melhor utilização da memória armazenando dados no número mínimo de bits exigido. Os membros dos campos de bits são declarados como “int” ou “unsigned”.

É possível especificar um campo de bits anônimo que é usado como enchimento (padding) na estrutura. Por exemplo, a “struct” abaixo usa um campo de bits anônimo de 3 bits como enchimento. O membro b é armazenado em outra unidade de armazenamento Um campo de bits anônimo com tamanho zero é usado para alinhar o próximo campo de bit no limite de uma nova unidade de armazenamento.

```
struct exemplo{
     unsigned a: 13;

     unsigned c: 3;

     unsigned b: 4;

};
```

7.Constantes de enumeração

A linguagem C fornece um tipo final definido pelo usuário chamado de uma enumeração. Uma enumeração, apresentada pela palavra-chave “enum”, é um conjunto de constantes inteiras representadas por identificadores. Essas constantes de enumeração são, na realidade, constantes simbólicas cujos valores podem ser definidos automaticamente. Os valores em um “enum” iniciam com 0, a menos que seja especificado de outra forma, e são incrementados de 1.