1.As classes de armazenamento

A linguagem C fornece quatro classes de armazenamento indicadas pelos especificadores de classes de armazenamento: ***auto***, ***register***, ***extern*** e ***static***. Uma classe de armazenamento de um identificador ajuda a determinar seu tempo de armazenamento, escopo e linkage. Um tempo de armazenamento de um identificador é o período durante o qual aquele identificador permanece na memória. Alguns identificadores permanecem pouco tempo, alguns são criados e eliminados repetidamente, e outros permanecem durante toda a execução do programa. O escopo de um identificador é onde se pode fazer referência àquele identificador dentro de um programa. Pode-se fazer referência a alguns identificadores ao longo de todo o programa, outros apenas a partir de determinados locais de um programa. A linkage de um identificador determina, para um programa com vários arquivos-fonte, se um identificador é conhecido apenas no arquivo-fonte atual ou em qualquer arquivo-fonte com as declarações adequadas.

As quatro classes de armazenamento dos identificadores podem ser divididas em dois tempos de armazenamento: ***tempo de armazenamento automático*** (auto e register) e ***tempo de armazenamento estático*** (extern e static).

As variáveis com tempo de armazenamento automático são criadas quando o bloco no qual são declaradas é adicionado, elas existem enquanto o bloco estiver ativo e são destruídas quando o bloco é deixado para trás. As variáveis locais de uma função possuem normalmente tempo de armazenamento automático.

A palavra-chave ***auto*** declara explicitamente variáveis de tempo de armazenamento automático. A declaração a seguir indica que as variáveis ***x*** e ***y*** do tipo ***float*** são variáveis locais e automáticas e existem apenas no corpo da função na qual a declaração acontece. As variáveis locais possuem tempo de armazenamento automático por *default*, portanto, a palavra-chave **auto** raramente é usada.

```
auto float x, y;
```

A palavra-chave ***register*** pode ser colocada antes de uma declaração de variável automática para indicar ao compilador que a variável seja conservada em um dos registradores de hardware de alta velocidade do computador. Se as variáveis forem usadas frequentemente como contadores ou para cálculo de totais, elas podem ser conservadas em registros de hardware, o overhead de carregar repetidamente as variáveis da memória nos registros e armazenar os resultados novamente na memória pode ser eliminado.

O compilador pode ignorar as declarações ***register***. Por exemplo, pode não haver número suficiente de registros disponíveis para o uso do compilador. A declaração a seguir indica que a variável inteira ***contador*** pode ser colocada em um

dos registros do computador e ser inicializada com 1. A palavra-chave ***register*** só pode ser usada com variáveis de tempo de armazenamento automático.

As palavras-chave ***extern*** e ***static*** são usadas para declarar identificadores para variáveis e funções de tempo de armazenamento estático. Os identificadores de tempo de armazenamento estático existem desde o instante em que o programa começou a ser executado. Para variáveis, o armazenamento é alocado e inicializado uma vez quando o programa começa a ser executado. Para funções, o nome da função existe quando o programa começa a ser executado. Muito embora as variáveis e os nomes das funções existam quando o programa começa a ser executado, isso não significa que esses identificadores podem ser usados em todo o programa.

Há dois tipos de identificadores com tempo de armazenamento estático: identificadores externos e variáveis locais declaradas com o especificador de classe de armazenamento ***static***. As variáveis globais e nomes de funções são pertencentes à classe de armazenamento ***extern*** por *default*. As variáveis globais são criadas colocando declarações de variáveis fora de qualquer definição de função, e conservam seus valores ao longo de toda a execução do programa. As referências a variáveis globais e funções podem ser feitas em qualquer função que venha após suas declarações ou definições no arquivo.

As variáveis locais declaradas com a palavra-chave ***static*** são conhecidas apenas na função na qual são definidas, mas, diferentemente das variáveis automáticas, as variáveis locais ***static*** conservam seus valores quando a função é encerrada. Na próxima vez em que a função for chamada, a variável local ***static*** conservará o valor que tinha quando a função foi executada pela última vez. Todas as variáveis numéricas de tempo de armazenamento estático são inicializadas com o valor zero se não forem inicializadas explicitamente pelo programador.

2.Regras de escopo

Um escopo de um identificador é a parte do programa na qual ele pode ser referenciado. Os quatro escopos de um identificador são: ***escopo de função***, ***escopo de arquivo***, ***escopo de bloco*** e ***escopo de protótipo de função***.

Os ***rótulos*** são os únicos identificadores com ***escopo de função***. Os rótulos podem ser usados em qualquer lugar de uma função na qual aparecem, mas não pode ser feita qualquer referência a eles fora do corpo da função. Os rótulos são usados em estrutura ***switch*** e em instruções ***goto***. Os rótulos são detalhes de implementação que funções ocultam de outras funções. Um identificador declarado fora de qualquer função tem escopo de arquivo. Tal identificador é conhecido por todas as funções desde o local onde é declarado até o final do arquivo.

Os **identificadores** declarados dentro de um bloco possuem **escopo de bloco**. O escopo de bloco termina na chave direita final do bloco. As variáveis locais

declaradas no início de uma função possuem escopo de bloco, assim como os parâmetros da função, que são considerados variáveis locais por ida. Qualquer bloco pode conter declarações de variáveis. Quando os blocos estão aninhados e um identificador em um bloco externo tem o mesmo nome de um identificador em um bloco interno, o identificador no bloco externo é “escondido” até que o bloco interno seja encerrado. Isso quer dizer que durante a execução de um bloco interno, este vê o valor de seu próprio identificador local, e não o valor identificador de mesmo nome no bloco externo.

As variáveis locais declaradas ***static*** também possuem escopo de bloco, muito embora existiam desde o instante em que o programa começou a ser executado. Dessa maneira, o tempo de armazenamento não afeta o escopo de um identificador.

Os únicos **identificadores** com ***escopo de protótipo de função*** são os utilizados na lista de parâmetros de um protótipo de função. Os protótipos de funções não necessitam de nomes em suas listas de parâmetros. O nome que for usado na lista de parâmetros de um protótipo de função será ignorado pelo compilador. Os identificadores usados em um protótipo de função podem ser reutilizados em qualquer lugar do programa sem ambiguidade.

3.Recursividade

Para alguns tipos de problemas, é útil ter funções que chamem a si mesmas. Uma ***função recursiva*** é uma função que chama a si mesma, ou diretamente ou indiretamente por meio de outra função. Os métodos recursivos para solução de problemas possuem vários elementos em comum. Uma função recursiva é chamada para resolver o problema. A função só sabe resolver caso(s) mais simples, ou o(s) chamado(s) caso(s) básico(s). Se a função for chamada em um problema mais complexo, ela divide o problema em duas partes teóricas: uma parte que a função sabe como resolver e outra que ela não sabe.

Para tornar viável a recursão, a segunda parte deve ser parecida com o problema original, mas ser uma versão um pouco mais simples ou menor do que ele. Por esse novo problema ser parecido com o problema original, a função chama uma nova cópia de si mesma para lidar com o problema menor. A etapa de recursão também inclui a palavra-chave ***return*** porque seu resultado será combinado com a parte do problema que a função sabe como resolver para formar um resultado que será enviado de volta para a função original de chamada, possivelmente ***main***.

A etapa de recursão é executada enquanto a chamada original para a função estiver ativa ou ainda não tiver sido concluída. A etapa de recursão pode levar a outras chamadas recursivas, à medida que a função continuar a dividir cada problema em duas partes conceituais. Para a recursão chegar ao fim, cada vez que a função chamar a si mesma com uma versão ligeiramente mais simples do problema original, essa

sequência de problemas cada vez menores deve convergir posteriormente para o caso básico. Nesse instante, a função reconhece o caso básico, envia um resultado de volta para a cópia anterior da função e ocorre uma sequência de remessa de resultados na ordem inversa até a chamada original da função enviar mais tarde o resultado para ***main***.