Resumo: Estudo da Linguagem C

Este resumo está em desenvolvimento, contando com atualizações sempre que me deparo com novas informações importantes a serem fixadas.

Sequencia de Escape:

\n	Nova linha. Posiciona o cursor no início da nova linha.
\t	Tabulação horizontal. Move o cursor para a próxima marca de tabulação.
\r	Carriage return (CR). Posiciona o cursor no início da linha atual.
\a	Alerta. Faz soar a campainha do sistema.
\\	Imprime o caractere de barra invertida (backslash) em uma instrução printf.
\”	Imprime o caractere de aspas duplas em uma instrução printf.

Operadores Aritméticos:

Adição	+
Subtração	-
Multiplicação	*
Divisão	/
Resto (Módulo)	%

Regras de Precedência:

Operador	Operação	Ordem de Cálculo (Precedência)
( )	Parênteses	Calculado em primeiro lugar. Se houver parênteses aninhados, a expressão dentro do parêntese mais interno é calculada em primeiro lugar. No caso de vários parênteses no mesmo nível, i.e., que não estejam aninhados, eles são calculados da esquerda para a direita.
* / %	Multiplicação Divisão Resto	Calculados em segundo lugar. No caso de vários operadores no mesmo nível, eles são calculados da esquerda para a direita.
+ -	Adição Subtração	Calculados por último. No caso de vários operadores no mesmo nível, eles são calculados da esquerda para a direita.

Operadores de Igualdade:

Igual	==
Não Igual (Diferente)	!=

Operadores Relacionais:

Maior	>
Menor	<
Maior ou Igual	>=
Menor ou Igual	<=

Operadores Condicionais:

? : intimamente relacionado com a estrutura if/else, porém podem ser usados em algumas situações que o if/else não podem.

Exemplos: printf(“%s\n”, nota >= 60 ? “Aprovado” : “Reprovado”);
nota >= 60 ? printf(“Aprovado\n”) : printf(“Reprovado\n”);

Associatividade:

( )	da esquerda para a direita
* / %	da esquerda para a direita
+ -	da esquerda para a direita
< <= > >=	da esquerda para a direita
== !=	da esquerda para a direita
=	da direita para a esquerda

Palavras Chave:

auto	break	case	Char
const	continue	default	do
double	else	enum	extern
float	for	goto	if
int	long	register	return
short	signed	sizeof	static
struct	switch	typedef	uniom
unsigned	void	volatile	while

Dicas retiradas do livro: Como Programar em C, de Paul J. Deitel e Harvey M. Deitel, 6ª Edição

Dicas de Portabilidade:

1. Como o C é uma linguagem independente de hardware e amplamente disponível, as aplicações escritas em C podem ser executadas com pouca ou nenhuma modificação em uma grande variedade de sistemas computacionais.
2. Usar as funções da biblioteca padrão do C em vez de escrever suas próprias versões similares pode melhorar a portabilidade do programa porque essas funções estão colocadas em praticamente todas as implementações do ANSI C.
3. Embora seja possível escrever programas portáteis, há muitos problemas entre as diferentes implementações do C e os diferentes computadores que tornam a portabilidade um objetivo difícil de atingir. Simplesmente escrever programas em C não garante a portabilidade.
4. Use identificadores com 31 caracteres ou menos. Isso ajuda a assegurar a portabilidade e pode evitar alguns erros sutis de programação.

Dicas de Desempenho:

1. Usar as funções da biblioteca stantard do C em vez de você escrever suas próprias versões similares por melhorar o desempenho do programa porque essas funções foram desenvolvidas cuidadosamente por pessoal eficiente.

Boas Práticas de Programação:

1. Escreva seus programas em C de uma maneira simples e objetiva. Algumas vezes isto é chamado KIS (“keep it simple”), mantenha a simplicidade. Não complique a linguagem tentando soluções estranhas.
2. Leias os manuais da versão do C que estiver usando. Consulte frequentemente estes manuais para se certificar do conhecimento do rico conjunto de recursos do C e de que eles estão sendo usados corretamente.
3. Seu computador e compilador são bons mestres. Se você não estiver certo de como funciona um recurso em C, escreva um programa de teste que utilize aquele recurso, compile e execute o programa, e veja o que acontece.
4. Todas as funções devem ser precedidas por um comentário descrevendo seu objetivo.
5. O último caractere impresso por uma função que realiza qualquer impressão deve ser o de nova linha (\n). Isto assegura que a função deixará o cursor da tela posicionado no início de uma nova linha. Procedimentos desta natureza estimulam a reutilização do software – um objetivo principal em ambientes de desenvolvimento de software.
6. Faça o recuo de um nível (três espaços) em todo o texto (corpo) de cada função dentro das chaves que a definem. Isto ressalta a estrutura funcional dos programas e ajuda a torná-los mais fáceis de ler.

7. Determine uma convenção para o tamanho de recuo preferido e então aplique-a uniformemente. A tecla de tabulação podem variar. Recomendamos usar paradas de tabulação de ¼ da polegada (aproximadamente 6 mm) ou recuar três espaços para cada nível de recuo.
8. Embora a inclusão de <stdio.h> seja opcional, ela deve ser feita em todos os programas em C que usam funções de entrada/saída da biblioteca padrão. Isto ajuda o compilador a localizar os erros na fase de compilação de seu programa em vez de na fase de execução (quando normalmente os erros são difíceis de corrigir).
9. Coloque um espaço depois de cada vírgula para tornar o programa mais legível.
10. Escolher nomes significativos para as variáveis ajuda a tornar um programa autoexplicativo, i.e., menos comentários se farão necessários.
11. A primeira letra de um identificador usado como nome de variável simples deve ser uma letra minúscula. Mais adiante no texto atribuiremos um significado especial aos identificadores que começam com uma letra maiúscula e aos identificadores que usam todas as letras maiúsculas.
12. Nomes de variáveis com mais de uma palavra podem ajudar a tornar o programa mais legível. Evite juntar as palavras separadas como em totalpagamentos. Em vez disso, separe as palavras com o sublinhado como em total_pagamentos ou, se você desejar juntar as palavras, comece cada palavra depois da primeira com uma letra maiúscula como em totalPagamentos.
13. Separe as declarações das instruções executáveis em uma função por uma linha em branco, para ressaltar onde terminam as declarações e começam as instruções.
14. Coloque espaços em ambos os lados de um operador binário. Isto faz com que o operador seja ressaltado e torna o programa mais legível.
15. Recue as instruções no corpo de uma estrutura if.
16. Coloque uma linha em branco antes e após todas as estruturas de controle em um programa para melhorar sua legibilidade.
17. Não deve haver mais de uma instrução por linha em um programa.
18. Colocar vírgulas (quando não são necessárias) entre os especificadores de conversão na string de controle de formato de de uma instrução scanf.
19. Uma instrução longa pode ser dividida em várias linhas. Se uma instrução deve ocupar mais de uma linha, escolha locais de divisão que façam sentido (como após uma vírgula em uma lista separada por vírgulas). Se uma instrução for dividida em duas ou mais linhas, recue todas as linhas subsequentes.
20. Consulte a tabela de precedência dos operadores ao escrever expressões que contenham muitos operadores. Certifique-se de que os operadores da expressão serão executados na ordem adequada. Se você não tiver certeza quanto à ordem de cálculo de uma expressão complexa, use parênteses para impor aquela ordem, exatamente do modo como é feito em expressões algébricas. Não se esqueça de levar em consideração que alguns operadores em C, como o operador de atribuição [=], são associados da direita para a esquerda e não da esquerda para a direita.
21. Aplicar consistentemente as convenções para recuos aumenta bastante a legibilidade do programa. Sugerimos valores fixos de aproximadamente ¼ da polegada ou três espaços em branco em cada nível de recuo.
22. Frequentemente, o pseudo código é usado para “elaborar” um programa durante o processo de projeto do programa. Depois o programa em pseudocódigo é convertido para o C.
23. Aplique recuos nas instruções em ambas as partes da estrutura if/else.
24. Se houver vários níveis de recuos, o recuo de cada nível deve ter a mesma quantidade adicional de espaços

Erros Comuns de Programação:

1. Esquecer de encerrar um comentário com */.
2. Começar um comentário com os caracteres */ ou terminar com /*.
3. Em um programa, digitar como print o nome da função de saída printf.
4. Usar uma letra maiúscula onde devia ser usada uma letra minúscula (por exemplo, digitar Main em vez de main).
5. Colocar declarações de variáveis entre instruções executáveis.
6. O cálculo de uma instrução de atribuição deve estar no lado direito do operador =. É um erro de sintaxe colocar o cálculo no lado esquerdo de um operador de atribuição.
7. Esquecer-se de uma ou de ambas as aspas duplas em torno de uma string de controle de formato de printf ou scanf.
8. Em uma especificação de conversão, esquecer-se do % na string de controle de formato de printf ou scanf.
9. Colocar uma sequencia de escape com \n fora da string de controle de formato de printf ou scanf.
10. Esquecer-se de incluir em uma instrução printf que contém especificadores de conversão as expressões cujos valores devem ser impressos.
11. Não fornecer um especificador de conversão para uma instrução printf, quando tal é exigido para imprimir uma expressão.
12. Colocar, dentro de uma string de controle de formato, a vírgula que deve separar a string de controle de formato das expressões a serem impressas.
13. Esquecer-se de preceder uma variável, em uma instrução scanf, de um e-comercial quando essa variável deve obrigatoriamente ser precedida por ele.
14. Preceder uma variável, incluída em uma instrução printf, de um e-comercial quando obrigatoriamente quela variável não deveria ser precedida por ele.
15. Normalmente, uma tentativa de dividir por zero não é definida em sistemas computacionais e em geral resulta em um erro fatal, i.e., um erro que faz com que o programa seja encerrado imediatamente sem ter sucesso na realização de sua tarefa. Erros não fatais permitem que os programas sejam executados até o final, produzindo frequentemente resultados incorretos.
16. Acontecerá um erro de sintaxe se os dois símbolos de qualquer um dos operadores ==, !=, >= e <= forem separados por espaços.
17. Acontecerá um erro de sintaxe se os dois símbolos em qualquer um dos operadores !=, >= e <= forem invertidos, como em =!, => e =<, respectivamente.
18. Confundir o operador de igualdade [==] com o operador de atribuição [=].
19. Colocar um ponto e vírgula imediatamente à direita do parêntese direito depois de uma condição em uma estrutura if.
20. Esquecer de uma ou ambas as chaves que limitam uma instrução composta.
21. Colocar um ponto e vírgula [;] depois da condição em uma estrutura if leva a um erro lógico em estruturas if de seleção simples e a um erro de sintaxe em estruturas if de seleção dupla.
22. Alguns programadores preferem digitar as chaves inicial e final de instruções compostas antes de digitar cada uma das instruções incluídas nessas chaves. Isso ajuda a evitar a omissão de uma ou ambas as chaves.

Observações de Engenharia de Software:

1. Uma instrução composta pode ser colocada em qualquer lugar de um programa no qual possa ser colocada uma instrução simples.

2. Da mesma forma que uma instrução composta pode ser colocada em qualquer local onde uma instrução simples pode estar, também é possível não ter instrução alguma, i.e., uma instrução vazia. A instrução vazia é representada colocando um ponto e vírgula [;] onde normalmente a instrução deveria estar.