

Introdução à linguagem C

Prof. Marcelo Costa, MSc
marcelo.nascimento@uva.edu.br

Agenda

- Histórico, definições e evolução;
- C versus C++;
- Compiladores versus Interpretadores;
- A forma de um programa C

Programação em Linguagem C

- Objetivos: apresentar uma visão geral da linguagem de programação C, suas origens, seus usos e sua filosofia

Programação em Linguagem C

- A linguagem C foi criada em um computador com SO Unix.
- É o resultado da evolução de um processo de desenvolvimento que começou com uma linguagem mais antiga, chamada BCPL, que influenciou a linguagem B, em 1970, que levou ao desenvolvimento da linguagem C.
- Desde 1983 utiliza padrão ANSI C (American National Standards Institute).

O que é C?

- É uma linguagem de Médio Nível*:
- Permite manipulação de bits, bytes e endereços de hardware;
- Portabilidade entre SOs;
- Não é uma linguagem rica em formatos básicos de dados: 5 formatos básicos;
- Permite quase todas as conversões de dados;
- Não realiza verificações em tempo de execução, devendo ser tratadas pelo programador.

Exemplo de programa em Linguagem C

```
#include <stdio.h>

#include <stdlib.h>

int soma(int a1, int a2){
    return(a1+a2);
}

int main(int argc, char *argv[])
{
    int A,B;

    scanf("%d",&A);
    scanf("%d",&B);
    printf("%d",soma(A,B));
    system("PAUSE");
    return 0;
}
```

Programação Estruturada

- Compartimentalização do código e dos dados: seccionar e esconder parte do código, ou modularização;
- Permite que os programas compartilhem facilmente seções de código;
- Permite a definição de escopo para variáveis;
- Permite a inserção de sentenças em qualquer parte de uma linha.

Programação Estruturada

- O principal componente estrutural de C é a função;
- O fato de se poder criar funções isoladas é extremamente importante em projetos maiores, nos quais um código de um programador não deve afetar acidentalmente o de outro.
- Possui a capacidade de construção de blocos de código

Exemplos de linguagens não-estruturadas

- COBOL
- Basic

Testada em campo por programadores profissionais:

- Características
 - Poucas restrições;
 - Poucas reclamações(warnings);
 - Estruturas de blocos;
 - Funções isoladas;
 - Conjunto compacto de palavras reservadas (32);
 - Pode ser utilizada como linguagem assembly (montagem), juntamente com o potencial de linguagens de maior nível.

Orientação Inicial

Inicialmente era utilizada para construção de softwares de sistema:

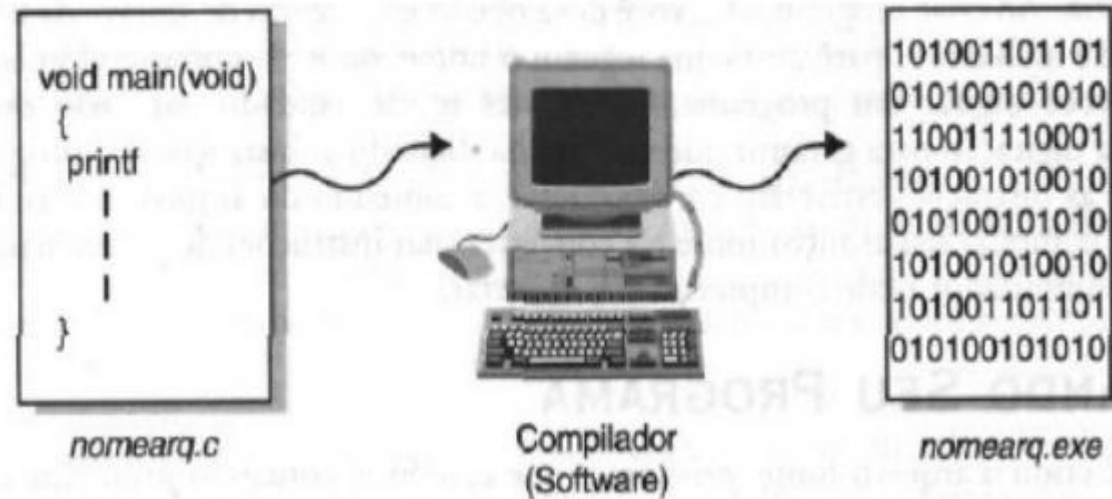
- Sistemas Operacionais;
- Interpretadores;
- Editores;
- Programas de planilhas eletrônicas;
- Compiladores;
- Gerenciadores de Bancos de Dados.

Linguagens Compiladas e Interpretadas

- Compiladores e interpretadores: “maneira como um programa é executado” (SCHILDT, 1995).
- Um interpretador lê o código-fonte linha a linha, executando a instrução contida em cada linha.
- Um compilador lê o programa inteiro e converte-o em um código-objeto, que é uma tradução do código-fonte em um formato que o sistema operacional possa executar diretamente.
- Quando um programa interpretado for rodar, ele requer a presença do interpretador, sempre que este for ser executado.

Compiladores e Interpretadores

Compiladores Vs Interpretadores



Estrutura de um programa em C

<Seção de importação de arquivos externos/diretivas>

<Seção de declarações>

<Corpo do programa>

Instruções em linguagem C:

- Instruções são comandos que o programa irá executar.
- Podem ser escritos em qualquer lugar do programa, mas devem ser:
 - Encerrados com ponto-vírgula (;)
 - Blocos de comandos são iniciados e encerrados com chaves: { }

Primeiro programa em C

```
#include <stdio.h>
#include <stdlib.h>
int main()
{
    printf("Primeiro programa");
    system("PAUSE");
    return 0;
}
```


Diretiva include

- `#include <stdio.h>` //são diretivas de compilação
- `#include <stdlib.h>` //que importam arquivos de cabeçalho
- A diretiva include provoca a inclusão de outro arquivo no programa fonte.
- É chamada de diretiva pre-processador, pois é uma instrução explícita ao Compilador para substituir o comando pelos dados existentes no arquivo.
- Cada header incluído possui funções diversas organizadas por conjunto comum de funcionalidades, como recursos de E/S.

Comentários em linguagem C

//comentário em linha

/*

Comentários em múltiplas linhas

*/

5 Tipos básicos de dados:

- char
- int
- float
- double
- void

Modificadores

Modificadores: signed, unsigned, short e long

Tipo	Tamanho aproximado em bits	Faixa mínima
char	8	-127 a 127
unsigned char	8	0 a 255
signed char	8	-127 a 127
int	16	-32.767 a 32.767
unsigned int	16	0 a 65.535
signed int	16	O mesmo que int
short int	16	O mesmo que int
unsigned short int	16	0 a 65.535
signed short int	16	O mesmo que short int
long int	32	-2.147.483.647 a 2.147.483.647
signed long int	32	O mesmo que long int.
unsigned long int	32	0 a 4.294.967.295
float	32	Seis dígitos de precisão
double	64	Dez dígitos de precisão
long double	80	Dez dígitos de precisão

Identificadores

- Nomes de variáveis, funções, rótulos e demais objetos definidos pelo programador são chamados de identificadores!
- Primeiro caractere sempre será uma letra, ou _;
- Após o primeiro caractere podem existir números;
- Nenhum outro caractere especial (ç~^!?!*%&%) além do _ pode ser utilizado;
- Não se separa palavras compostas;

Exemplos de Identificadores

Corretos	Incorretos
Count	1Count
Preco	1_Count
_Valor_1	valor 1
count1	Preço

Boas práticas

- nomes_de_variaveis_extremamente_longos_podem_ser_ruins
- Não se utiliza o mesmo identificador de uma função, e não se utiliza palavras reservadas da linguagem.
- **Case Sensitive**
 - A linguagem C reconhece caracteres maiúsculos como diferentes dos caracteres minúsculos:
 - Count ≠ count ≠ COUNT

Declaração de Variáveis

- Sintaxe:
 <tipo_de_dados> <lista_de_variaveis>;
- Ex.:
 - int i,j, L;
 - double profit, balance, loss;

Declaração de variáveis – ONDE DECLARAR?

- Dentro de funções: variáveis locais
- Na definição de parâmetros das funções: parâmetros formais
- Fora de qualquer função: variáveis globais

Declaração de variáveis – ONDE DECLARAR?

- ex.:
...
int i,j;
int main(int a){
 int valor1;
 valor1 = 1;
 return(1);
}

Variáveis Globais (fora do main()) – acessado de qualquer lugar)

Definição de Parâmetros formais

Variáveis Locais

Obs.: variáveis locais são destruídas após a saída do bloco.

Inicialização de Variáveis

- Inicializar uma variável é preenchê-la com um valor inicial.
- Em C, pode-se dar um valor inicial à uma variável imediatamente após criá-la.

- Sintaxe:

`<tipo> <identificador> = <constante>;`

ex.:

`char ch = 'a';`

`int valor = 10;`

`float preco = 10.99;`

Atribuição de valores em C

- Em C, constantes referem-se a valores fixos que o programa não pode alterar.
- Podem ser de qualquer um dos 5 tipos básicos de dados.
- Constante de caractere: envolvidas por aspas simples ('')

Ex.: 'a'

- Constante inteira: números sem fração

Ex.: 10

- Constante de ponto flutuante: números com fração

Ex.: 1.99

- Constante hexadecimal ou octal

Ex.: 0x80 //128 em decimal

- Constante de cadeia de caracteres: string

Conjunto de caracteres entre aspas duplas;

Ex.: “isso é uma cadeia”

Operadores em C

- A linguagem C é muito rica em operadores;
- Operadores tradicionais:
 - Aritméticos
 - Relacionais
 - Lógicos
 - Bit a Bit

Operador de atribuição:

- Sintaxe: <identificador> = <valor>
 - Exemplos:
 - Preço = 1.01;

Operador de atribuição:

- Atribuição múltipla:
 - Exemplos:
 - $A = B = C = D = 1;$

Conversão excplícita de tipos em atribuição

Tabela 2.3 Conversões de tipos comuns (assumindo uma palavra de 16 bits).

Tipo do destino	Tipo da expressão	Possível informação perdida
signed char	char	Se valor > 127, o destino é negativo
char	short int	Os 8 bits mais significativos
char	int	Os 8 bits mais significativos
char	long int	Os 24 bits mais significativos
int	long int	Os 16 bits mais significativos
int	float	A parte fracionária e possivelmente mais
float	double	Precisão, o resultado é arredondado
double	long double	Precisão, o resultado é arredondado

Exemplos

```
int total, contador;
```

```
float media;
```

```
media = (float) total / contador;
```

- O resultado de `total / contador` é inteiro porque `total` e `contador` são valores inteiros. Dividir dois inteiros resulta em uma divisão inteira na qual a parte fracionária é ignorada antes de o resultado ser atribuído a `media`.
- A linguagem C fornece o operador unário de coerção (conversão) para realizar essa tarefa no caso acima o `float` para o `total`.
- O C promove o `contador` temporariamente para `float` para conseguir realizar a operação. (Promoção de Tipos)

Operadores Aritméticos

Tabela 2.4 Operadores aritméticos.

Operador	Ação
-	Subtração, também menos unário
+	Adição
*	Multiplicação
/	Divisão
%	Módulo da divisão (resto)
--	Decremento
++	Incremento

Operadores aritméticos:

- Incremento e decremento de variáveis
 - $X = X + 1;$
- É o mesmo que
 - $++X;$ ou $X++$
- $X = X - 1;$
 - É o mesmo que
 - $--X;$ ou $X--;$

Operadores Relacionais

Operadores relacionais	
Operador	Ação
>	Maior que
>=	Maior que ou igual
<	Menor que
<=	Menor que ou igual
==	Igual
!=	Diferente

Álgebra: $m = \frac{(a + b + c + d + e)}{5}$

C: $m = (a + b + c + d + e) / 5;$

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

O próximo exemplo é a equação de uma reta:

Álgebra: $y = mx + b;$

C: $y = m * x + b;$

Operadores Lógicos

Operadores lógicos			
	Operador	Ação	
	&&	AND	
	!!	OR	
	!	NOT	
<hr/>			
q	p&&q	p!!q	!p
0	0	0	1
1	0	1	1
1	1	1	0
0	0	1	0

Operadores lógicos e relacionais:

- `10>5 // Verdadeiro`
- `2<=1 // Falso`
`1>2 && 1>0 // Falso`
- A. `10>5 && !(10<9) || 3 <=4 // Verdadeiro`

Operadores ternário ?:

- C contém um operador muito poderoso e conveniente, o operador ternário ?, com a sintaxe:
- $\text{Exp1} ? \text{Exp2} : \text{Exp3}$
- Onde Exp1 é avaliada, sendo verdadeira, Exp2 é avaliada, se Exp1 for falso, Exp3 será avaliada.
- ex.:
x = 10;
y = x > 9 ? 100 : 200; // y = 100

Operador C reduzido:

- Variante do comando de atribuição `=`, que simplifica algumas operações de atribuição:
 - `X = X+10;`
- Pode ser escrito como:
 - `X += 10;`
 - `X = X - 100;`
- Pode ser escrito como:
 - `X -= 100;`

Entrada e saída de dados em um programa em linguagem C

A função printf()

- A instrução `printf("primeiro programa");`
- É a chamada da função `printf()`, que dará saída no monitor da cadeia de caracteres
- `"primeiro programa"`.
- É uma função com argumento textual.

função printf()

- Além do ENTER, diversos outros caracteres não podem ser inseridos em
- Uma cadeia para ser apresentada pela função printf(). Para contornar
- isso, faz-se uso de caracteres de escape:
- \n - nova linha
- \t - tabulação
- \\ - barra invertida
- \0 - numeral zero
- \' - aspas simples
- \" - aspas duplas

função printf()

- Para a correta saída formatada dos dados, se faz necessário a utilização de modificadores:
 - %c- Caractere simples
 - %d - Inteiro decimal com sinal
 - %u - Inteiro decimal sem sinal
 - %f - decimal
 - %s- Cadeia de caracteres
 - %o - Inteiro octal sem sinal
 - %x- Inteiro hexadecimal sem sinal
 - %X - Inteiro hexadecimal sem sinal
 - Coloca-se o l antes da letra pra indicar que é long

A função printf()

- Ex.:

```
#include <stdio.h> //para usar printf
#include <stdlib.h> //para usar system
int main(){
    printf("Este é o número %d. \n", 5);
    printf("%s esta a %d milhões de milhas\n do sol.\n", "Venus", 67);
    System("PAUSE");
    Return 0;
}
```

Explorando a função printf():

- A função printf() usa uma frase (cadeia de caracteres, ou simplesmente string) para escrever na tela um resultado ou texto desejado.
- printf("Saída do programa");
- Para imprimirmos um valor constante ou o valor de uma variável, utiliza-se o símbolo % seguido de uma letra, que será o formatador de saída.
- printf("O resultado é:%d", 10);

Explorando a função printf():


- A função printf() permite estabelecer o tamanho mínimo de um campo para impressão.
- Isso significa que poderemos definir o nº de colunas que serão ocupadas por um valor a ser impresso. Muito útil em controle de formulários e tabelas:

```
/* Tamanho de campo com inteiros */  
#include <stdio.h> /* Para printf() */  
#include <stdlib.h> /* Para system() */  
int main()  
{  
    printf("Os alunos sao %2d.\n",350);  
    printf("Os alunos sao %4d.\n",350);  
    printf("Os alunos sao %5d.\n",350);  
    system("PAUSE");  
    return 0;  
}
```


Explorando a função printf()

Quantidade de espaços antes da impressão

```
int main() {  
    int lapis=45, borrachas=2345, canetas=420, cadernos=8,fitas=13050;  
    printf("\nLapis %12d",lapis);  
    printf("\nBorrachas %12d",borrachas);  
    printf("\nCanetas %12d",canetas);  
    printf("\nCadernos %12d",cadernos);  
    printf("\nFitas %12d",fitas);  
    system("PAUSE");  
    Return 0;  
}
```



Explorando a função printf():

Pode-se obter precisão de arredondamento de ponto flutuante.

```
int main() {  
    float lapis=4.785, borrachas=234.542, canetas=42.036, cadernos=8.0,  
    fitas=13.050;  
    printf("\nLapis %12.2f",lapis);  
    printf("\nBorrachas %12.2f",borrachas);  
    printf("\nCanetas %12.2f",canetas);  
    printf("\nCadernos %12.2f",cadernos);  
    printf("\nFitas %12.2f",fitas);  
    system("PAUSE");  
    Return 0;  
}
```

Quantidade de espaços antes da impressão

Quantidade de casas decimais

Completando com zeros à esquerda:

```
int main() {  
    //completando com zeros a esquerda  
    printf("\n\n");  
    printf("\n%04d",21);  
    printf("\n%06d",21);  
    printf("\n%6.4d",21);  
    printf("\n%6.0d",21);  
    system("PAUSE");  
    Return 0;  
}
```

Escolhendo a base numérica de saída da informação:

- ```
Eint main() {
 //alterando a base numerica da saida
 printf("\n\n");
 printf("\n%d",65);//decimal
 printf("\n%x",65);//hexadecimal
 printf("\n%o",65);//octal
 printf("\n%c",65);//character
 system("PAUSE");
 Return 0;
}
```

## Exercício

- Desenvolva um programa em linguagem C para escrever uma tabela de produtos e preços, bem como suas descrições.
- Liste 5 produtos. Utilize variáveis para os preços dos produtos.

## Lendo dados de entrada do teclado

- A função `scanf()` é outra função I/O presente na biblioteca padrão da linguagem C, fornecida com os compiladores C. Está definida em **`stdio.h`**.
- Ela é complemento da função `printf()` e nos permite ler dados da entrada padrão (teclado).
- Sintaxe:  
`scanf("expressão de controle", lista de argumentos);`  
A expressão de controle deve conter formatadores, e a lista de argumentos deve conter a(s) variável(eis) onde serão guardados os valores entrados.

| Códigos de formatação para scanf() | Significado                            |
|------------------------------------|----------------------------------------|
| %c                                 | Caractere simples.                     |
| %d                                 | Inteiro decimal com sinal.             |
| %i                                 | Inteiro decimal, hexadecimal ou octal. |
| %e                                 | Notação científica.                    |
| %f                                 | Ponto flutuante em decimal.            |
| %g                                 | Usa %e ou %f, o que for menor.         |
| %o                                 | Inteiro octal.                         |
| %s                                 | String de caracteres.                  |
| %u                                 | Inteiro decimal sem sinal.             |
| %x                                 | Inteiro hexadecimal.                   |
| %ld                                | Inteiro decimal longo.                 |
| %lf                                | Ponto flutuante longo (double).        |

## Explorando a função scanf()

```
int main() {
 int anos, dias;
 printf("\nDigite sua idade em anos: ");
 scanf("%d",&anos);
 dias = anos * 365;
 printf("\nSua idade em dias e:%d",dias);
 system("PAUSE");
 Return 0;
}
```



# Explorando a função scanf():

Múltiplas entradas com scanf().:

```
int main() {
 float p1, p2, p3, p4;
 double media;
 printf("\nDigite as notas das 4 provas: ");
 scanf("%f%f%f%f",&p1, &p2, &p3, &p4);
 media = (p1+p2+p3+p4)/4.0;
 printf("\nMedia: %4.2f\n",media);
 system("PAUSE");
 Return 0;
}
```

## Exercício

- Desenvolva um programa em linguagem C para LER uma tabela de produtos e preços, bem como suas descrições.
- Liste 5 produtos. Utilize variáveis para os preços dos produtos.

## Explorando a função scanf():

- Faça um programa em C, para ler a carga horária de um curso e a quantidade de faltas de um aluno, considerando em horas.
- Calcule e apresente a porcentagem de faltas que este aluno possui.

## Explorando a função getchar()

- A função getchar() é uma alternativa ao scanf, na sintaxe utilizada abaixo:

```
char ch;
```

```
printf("\nDigite uma tecla");
```

```
ch = getchar(); //aguarda uma tecla do teclado
```

```
printf("\nA tecla digitada ASCII: %c.\n", ch);
```

Presente em stdio.h

## Explorando a função getch()

- A função getch() retorna o caractere digitado, sem a necessidade de aguardar que se tecele o enter.

```
char ch;
printf("\nDigite uma tecla");
ch = getch(); //aguarda uma tecla do teclado
printf("\nA tecla digitada ASCII: %c.\n", ch);
```

Presente em conio.h

## Explorando a função putchar()

- A função getch() retorna o caractere digitado, sem a necessidade de aguardar que se tecele o enter.

```
char ch;
```

```
printf("\nDigite uma tecla");
```

```
ch = getch(); //aguarda uma tecla do teclado
```

```
printf("\nA tecla digitada ASCII: %c.\n");
```

```
putchar(ch);
```

```
//putchar('c');
```

```
Presente em stdio.h
```

# Explorando incremento e decremento

- O uso do incremento (++) opera sobre o valor de uma variável inteira e adiciona 1 ao seu valor. Pode operar de forma préfixada ou pós-fixada.

```
Int n = 5, x;
```

```
x = ++n;
```

```
printf("\nN:%d – X:%d",n,x);
```

```
// a saída será N:6 – X:6
```

```
int n = 5, x;
```

```
x = n++;
```

```
printf("\nN:%d – X:%d",n,x);
```

```
// a saída será N:6 – X:5
```

# Operadores aritméticos de atribuição

- São operadores que combinam os operadores aritméticos com um operador de atribuição:  $+=$   $-=$   $*=$   $/=$

$i += 2$ ; equivale a  $i = i + 2$ ;

$i *= y + 1$ ; equivale a  $i = i * (y + 1)$ ;

$i /= 2$ ; equivale a  $i = i / 2$ ;

$i -= 2$ ; equivale a  $i = i - 2$ ;



## E o tipo de dados lógico?

- Em C não há um tipo de dados booleano, que aceita verdadeiro(true) ou falso(false). Para isso, assume-se que falso vale 0 e verdadeiro vale 1.

# Estruturas de controle em C

# Agenda

- Seleção
- Iteração
- Bloco
- Laços

# Seleção simples e composta

- Definem fluxo condicional de execução do algoritmo.

Sintaxe:

if (expressão lógica) comando;

else comando;

if (expressão lógica) {

A.        comandos

B.    }else{

C.        comandos

D.    }

## Seleção simples e composta

```
void main(void)
{
 int magic; /* número mágico */
 int guess; /* palpite do usuário */

 magic = rand(); /* gera o número mágico */

 printf("adivinha o número mágico: ");
 scanf("%d", &guess);

 if(guess == magic) printf("*** Certo ***");
 else printf("Errado");
}
```

# Ifs aninhados

- Um if aninhado é um if que é objeto de outro if ou else. Em C, um comando else sempre se refere ao comando if mais próximo.

Exemplo:

```
if (i){
 if (j) comando1;
 if (k) comando2 //esse if
 else comando3; //está associado a este else
} else comando4; //este else está associado a if (i)
```

```
void main(void)
{
 int magic; /* número mágico */
 int guess; /* palpite do usuário */

 magic = rand(); /* gera o número mágico */

 printf("Adivinhe o número mágico: ");
 scanf("%d", &guess);

 if(guess == magic) {
 printf("*** Certo ***");
 printf(" %d é o número mágico\n", magic);
 }
 else {
 printf("Errado, ");
 if(guess > magic) printf("muito alto\n");
 else printf("muito baixo\n");
 }
}
```

? como alternativa à if/else:

```
int x = 10;
```

```
if (x>9) y = 100;
```

```
else y = 200;
```

```
int x = 10;
```

```
y = x>9 ? 100 : 200;
```



# Operador ternário aninhado

```
int x = 10;
```

```
if (x>9) {
```

```
 if (x==10) y=100;
```

```
 else y=10;
```

```
}else y = 200;
```

```
int x = 11;
```

```
int y;
```

```
y = x>9 ? (x==10 ? 100 : 10) : 200;
```

# Múltiplas seleções com switch:

- A linguagem C tem um comando de seleção múltipla, o switch, que testa sucessivamente o valor de uma expressão contra uma lista de constantes inteiras ou de caractere. Quando o valor coincide, os comandos associados à constante são executados. Sintaxe:
- ```
switch(expressão){  
    case constante1:  
        comandos;  
        break;  
    case constante2:  
        comandos;  
        break;  
    ...  
    default:  
        comandos;  
}
```

```
char ch;

printf("1. Checar Ortografia\n");
printf("2. Corrigir Erros de Ortografia\n");
printf("3. Mostrar Erros de Ortografia\n");
printf("Pressione Qualquer Outra Tecla para Abandonar\n");
printf("      Entre com sua escolha:  ");

ch=getchar(); /* Lê do teclado a seleção */

switch(ch) {
    case '1':
        check_spelling();
        break;
    case '2':
        correct_errors();
        break;
    case '3':
        display_errors();
        break;
    default :
        printf("Nenhuma opção selecionada");
}
```

```
int flag;
```

```
flag = -1;
```

```
switch(i) {
```

```
    case 1: /* Estes cases têm uma sequência */
```

```
    case 2: /* de comandos em comum */
```

```
    case 3:
```

```
        flag = 0;
```

```
        break;
```

```
    case 4:
```

```
        flag = 1;
```

```
    case 5:
```

```
        error(flag);
```

```
        break;
```

```
    default:
```

```
        process(i);
```

```
}
```

Switch aninhado

```
switch(x) {  
    case 1:  
        switch(y) {  
            case 0: printf ("erro de divisão por zero");  
                    break;  
            case 1: process(x,y);  
        }  
        break;  
    case 2:  
        .  
        .  
        .  
}
```

Laços de repetição

- Relembrando que um laço de repetição garante a repetição de comandos sob determinadas condições.
- O laço for:
sintaxe:
`for (inicialização; condição; incremento) comando;`
- O laço for permite muitas variações, mas é comum ter a inicialização com um comando de atribuição, que coloca um valor inicial para o passo. A condição é uma expressão condicional que garante o início e fim do laço, e incremento garante o passo.

Laços de repetição

```
#include <stdio.h>

void main(void)
{
    int x;

    for(x=1; x <= 100; x++) printf("%d ",x);
}
```

Laços de repetição

- Se existirem múltiplos comandos, um bloco deve ser criado

```
x = 10;  
for(y=10; y!=x; ++y) printf("%d", y);  
printf("%d", y); /* este é o único comando  
                  printf() que executará */
```

```
for(x=100; x != 65; x-=5) {  
    z = x*x;  
    printf("O quadrado de %d, %f", x, z);  
}
```


Laços de repetição

```
for(x=0, y=0; x+y<10; ++x) {  
    y = getchar();  
    y = y-'0'; /* subtrai o código ASCII do caractere 0 de y */  
    .  
    .  
    .  
}
```

Laços de repetição- o laço While

- É um laço com teste no início.
- Outro laço disponível em C é o laço while, com a sintaxe:

while (condição) comando;

char ch;

ch = '';

while (ch != 'A') ch = getchar();

while ((ch=getchar()) != 'A'); //while sem corpo

Laços de repetição- o laço While

- While com múltiplos comandos utiliza-se um bloco de comandos:

```
int working;

working = 1;  /* i.e., verdadeiro */

while(working) {
    working = process1();
    if(working)
        working = process2();
    if(working)
        working = process3();
}
```

Laços de repetição- o laço do-While

- Com teste no fim.
- Ao contrário dos laços for e while, que testam no início, o laço do-while verifica a condição ao final do laço. Isso significa que um laço do-while sempre será executado ao menos uma vez.
- Sintaxe:
do{
comando;
}while(condição)

```
do {  
    scanf("%d", &num);  
} while(num > 100);
```

```
void menu(void)
{
    char ch;

    printf("1. Verificar Ortografia\n");
    printf("2. Corrigir Erros de Ortografia\n");
    printf("3. Mostrar Erros de Ortografia\n");
    printf("      Digite sua escolha:  ");

    do {
        ch = getchar(); /* lê do teclado a seleção */
        switch(ch) {
            case '1':
                check_spelling();
                break;

            case '2':
                correct_errors();
                break;
            case '3':
                display_errors();
                break;
        }
    } while(ch!='1' && ch!='2' && ch!='3');
}
```