Introdução à linguagem C

Diego Raphael Amancio

Baseado no material do Prof. Thiago A. S. Pardo e do Prof. André Backes

Comandos de Seleção

- São também chamados de comandos condicionais.
  - if
  - switch

Comando if

- Forma geral:

if (*expressão***)** *sentença1***;**

else *sentença2***;**

- *sentença1* e *sentença2* podem ser uma única sentença, um bloco de sentenças, ou nada.
- O **else** é opcional.

Comando if

- Se *expressão* é verdadeira (!= 0), a sentença seguinte é executada. Caso contrário, a sentença do **else** é executada.

- O uso de if-else garante que apenas uma das sentenças será executada.

Comando if -- ambiguidades

```
if ( 1 /*true*/ )
    if ( 0  /*false*/ )
        comando1;
else
        comando2;
```

Não executa nenhum comando ?

Comando if

```
if ( 1 /*true*/ )
{
    if ( 0 /*false*/ )
        comando1;
   else
        comando2;
}
```

Executa comando2

Comando if

- comando if pode ser aninhado.
  - Possui em sentença um outro if.
  - ANSI C especifica máximo de 15 níveis.

- **Cuidado**: um else se refere, sempre, ao if mais próximo, que está dentro do mesmo bloco do else e não está associado a outro if.

Comando Switch

- **switch** (expressão) {
  case *constante1*: *sequência1*; **break**;
  case *constante2*: *seqüência2*; **break**;
  ...
  default: *seqüência_n*;
  }

Comando Switch - cuidados

- Testa a igualdade do valor da expressão com constantes somente.
- Duas constantes case no mesmo switch não podem ter valores idênticos.
- Se constantes caractere são usadas em um switch, elas são automaticamente convertidas em inteiros
- **break** é opcional.
- **default** é opcional.
- **switch** pode ser aninhado.

Exemplo

```
int x;
scanf(“%d”, &x);
switch (x) {
    case 1: printf(“Um”); break;
    case 2: printf(“Dois”); break;
    case 3: printf(“Tres”); break;
    case 4: printf(“Quatro”); break;
    default: printf(“ default “);
```

Exemplo

```
int x;
scanf("%d", &x);
switch (x) {
    case 1: printf("Um"); break;
    case 2: printf("Dois"); break;
    case 3: printf("Tres"); break;
    case 4: printf("Quatro"); break;
    default: printf("  ");
```

O que acontece se o primeiro *break* for removido?

Comandos de Iteração

- Comando for
- Comando while
- Comando do-while

Comando for

- **for (***inicialização* **;** *condição* **;** *incremento***)** comando**;**
- As três seções – inicialização, condição e incremento - devem ser separadas por ponto-e-vírgula (;)
- Quando condição se torna falsa, programa continua execução na sentença seguinte ao for.

Exemplo-for

```
#include <stdio.h>
int main (void) {
   int i;
   for (i=0; i<10; i++)
        printf("%d \n", i);

   return(0);
}
```

1. Inicialização i = 0
2. Teste i < 10
3. Escreve 0
4. i++
5. Teste i <10
6. Escreve 1
7. i++
8. ...
9. Escreve 9
10. i++
11. Teste i < 10
12. return 0

Exemplo-for

```
#include <stdio.h>
int main (void) {
   int i;
   for (i=0; i<10; i++)
        printf("%d \n", i);

   return(0);
}
```

1. Inicialização i = 0
2. Teste i < 10
3. Escreve 0
4. i++
5. Teste i <10
6. Escreve 1
7. i++
8. ...
9. Escreve 9
10. i++
11. Teste i < 10
12. return 0

Exemplo-for

```
#include <stdio.h>
int main (void) {
   int i;
   for (i=0; i<10; i++)
        printf("%d \n", i);

   return(0);
}
```

1. Inicialização i = 0
2. Teste i < 10
3. Escreve 0
4. i++
5. Teste i <10
6. Escreve 1
7. i++
8. ...
9. Escreve 9
10. i++
11. Teste i < 10
12. return 0

Exemplo-for

```
#include <stdio.h>
int main (void) {
   int i;
   for (i=0; i<10; i++)
        printf("%d \n", i);

   return(0);
}
```

1. Inicialização i = 0
2. Teste i < 10
3. Escreve 0
4. i++
5. Teste i <10
6. Escreve 1
7. i++
8. ...
9. Escreve 9
10. i++
11. Teste i < 10
12. return 0

Exemplo-for

```
#include <stdio.h>
int main (void) {
   int i;
   for (i=0; i<10; i++)
        printf("%d \n", i);

   return(0);
}
```

1. Inicialização i = 0
2. Teste i < 10
3. Escreve 0
4. i++
5. Teste i <10
6. Escreve 1
7. i++
8. ...
9. Escreve 9
10. i++
11. Teste i < 10
12. return 0

Exemplo-for

```
#include <stdio.h>
int main (void) {
   int i;
   for (i=0; i<10; i++)
        printf("%d \n", i);

   return(0);
}
```

1. Inicialização i = 0
2. Teste i < 10
3. Escreve 0
4. i++
5. Teste i <10
6. Escreve 1
7. i++
8. ...
9. Escreve 9
10. i++
11. Teste i < 10
12. return 0

Exemplo-for

```
#include <stdio.h>
int main (void) {
   int i;
   for (i=0; i<10; i++)
        printf("%d \n", i);

   return(0);
}
```

1. Inicialização i = 0
2. Teste i < 10
3. Escreve 0
4. i++
5. Teste i <10
6. Escreve 1
7. i++
8. ...
9. Escreve 9
10. i++
11. Teste i < 10
12. return 0

Exemplo-for

```
#include <stdio.h>
int main (void) {
   int i;
   for (i=0; i<10; i++)
        printf("%d \n", i);

   return(0);
}
```

1. Inicialização i = 0
2. Teste i < 10
3. Escreve 0
4. i++
5. Teste i <10
6. Escreve 1
7. i++
8. ...
9. Escreve 9
10. i++
11. Teste i < 10
12. return 0

Exemplo-for

```
#include <stdio.h>
int main (void) {
   int i;
   for (i=0; i<10; i++)
        printf("%d \n", i);

   return(0);
}
```

1. Inicialização i = 0
2. Teste i < 10
3. Escreve 0
4. i++
5. Teste i <10
6. Escreve 1
7. i++
8. ...
9. Escreve 9
10. i++
11. Teste i < 10
12. return 0

Exemplo-for

```
#include <stdio.h>
int main (void) {
   int i;
   for (i=0; i<10; i++)
        printf("%d \n", i);

   return(0);
}
```

1. Inicialização i = 0
2. Teste i < 10
3. Escreve 0
4. i++
5. Teste i <10
6. Escreve 1
7. i++
8. ...
9. Escreve 9
10. i++
11. Teste i < 10
12. return 0

Exemplo-for

```
#include <stdio.h>
int main (void) {
   int i;
   for (i=0; i<10; i++)
        printf("%d \n", i);

   return(0);
}
```

1. Inicialização i = 0
2. Teste i < 10
3. Escreve 0
4. i++
5. Teste i <10
6. Escreve 1
7. i++
8. ...
9. Escreve 9
10. i++
11. Teste i < 10
12. return 0

i = 10

Exemplo-for

```
#include <stdio.h>
int main (void) {
   int i;
   for (i=0; i<10; i++)
        printf("%d \n", i);

   return(0);
}
```

1. Inicialização i = 0
2. Teste i < 10
3. Escreve 0
4. i++
5. Teste i <10
6. Escreve 1
7. i++
8. ...
9. Escreve 9
10. i++
11. Teste i < 10
12. return 0

Comando for

- As expressões (inicialização, condição, incremento e comando) são opcionais!

```
int x;
for (x = 0; x != 34; )
   scanf(“%d”, &x);
```

O que acontece?

Comando while

- Forma geral

  while (*condição***)**

  *comando***;**

- *condição*: é qualquer expressão. Determina o fim do laço: quando a condição é falsa.
- *comando*: pode ser vazio, simples ou um bloco.

Exemplo- while

```
#include<stdio.h>
void main(){
        int x;
        scanf(“%d”,&x);
        while (x != -1)
                scanf(“%d”,&x);
}
```

Comando do-while

- Forma geral

do{

comando ;

} while (*condição***);**

- *comando*: pode ser vazio, simples ou um bloco.
- *condição*: pode ser qualquer expressão. Se falsa, o comando é terminado e a execução continua na sentença seguinte ao do-while

Comandos de desvio

- Comando return
- Comando break

Comando return

- Forma geral:

  return *expressão***;**

Comando break

- Dois usos:
  - Terminar um case em um comando switch.

```
int x;
switch(x){
    case 1: printf (“1”);
    case 2: printf (“2”);
    case 3: printf (“3”); break;
    case 4: printf (“4”);
    case 5: printf (“5”); break;
}
```

x = 1 → “123”
x = 2 → “23”
x = 3 → “3”
x = 4 → “45”
x = 5 → “5”

Comando break

- Forçar a terminação imediata de um laço

```
int x;
for ( x = 1; x < 100; x++ )
{
    printf( “%d ”, x );
    if ( x == 13 )
      break;
}
```

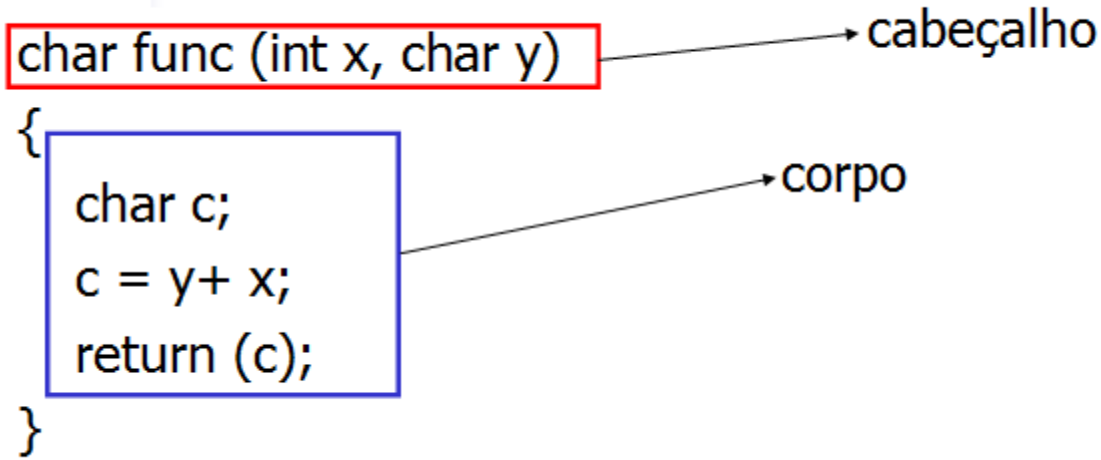
Funções -- declaração

- Forma Geral:

  tipo nome_da_função (lista de parâmetros)
  {declarações sentenças}

  - Tudo antes do “abre-chaves” compreende o **cabeçalho** da **definição** da função.
  - Tudo entre as chaves compreende o **corpo** da **definição** da função.

Exemplo- função



Funções

- *tipo nome_da_função (lista de parâmetros) {declarações sentenças}*
- **tipo:** é o tipo da função, i.e., especifica o tipo do valor que a função deve retornar (*return*).
  - Pode ser qualquer tipo válido.
  - Se a função não retorna valores, seu tipo é **void**.
  - Se o tipo não é especificado, tipo *default* é **int**.
  - Se necessário, o valor retornado será convertido para o tipo da função.

Exemplo

```
int suma(int x,int y);

void main() {
    int x,y, result;
    result= suma(x,y);
}

 int suma (int x, int y) {
      int c;
      c = y + x;
      return (c);
}
```

Passagem por valor

- Modo default de passagem em C
- Na chamada da função, os parâmetros são copiados localmente
- Uma mudança interna não acarreta uma mudança externa

```
void potencia2_valor (int n) { n = n * n; }
```

Passagem por valor

```
void potencia2_valor (int n) { n = n * n; }
```

Chamada:

```
int x = 3;
potencia2_valor ( x );
printf( “%d\n”, x );     //Imprime 3
```

Passagem por referência

- O endereço é passado na chamada da função
- Uma mudança interna acarreta uma mudança externa

```
void potencia2_ref (int *n) { *n = *n * *n ; }
```

Passagem por referência

```
void potencia2_ref (int *n) { *n = *n * *n ; }
```

Chamada:

```
int x = 3;
potencia2_ref ( &x );  //Chamada com endereço
printf( “%d\n”, x );     //Imprime 9
```

Passagem de matrizes

- É sempre feita por referência! Por quê?

```
void sort ( int num [ 10 ] );
void sort ( int num [ ] );
void sort ( int *num );
```

argc e argv

- Passando informações quando o programa é executado via linha de comando
- argc – número de argumentos passados
  - argc > 0 porque o nome do programa é o primeiro argumento
- argv: lista de argumentos passados
  - argv[0] – nome do programa
  - argv[1] – primeiro argumento ...

argc e argv

- argv – vetor de strings

```
void main(int argc, char *argv[])
{
    if ( argc != 2 )
        printf("Você esqueceu o segundo argumento")

    print( "Ola %s\n", argv[1]);
}
```

Chamada: ./program.exe myName

Tipos de funções

- As funções são geralmente de dois tipos
  1. Executam um cálculo: sqrt(), sin()
  2. Manipula informações: devolve sucesso/falha
  3. Procedimentos: exit(), retornam void

Não é necessário utilizar os valores retornados

Funções de tipo não inteiro

- Antes de ser usada, seu tipo deve ser declarado
- Duas formas
  - Método tradicional
  - Protótipos

Funções de tipo não inteiro

- Método tradicional
  - Declaração do tipo e nome antes do uso
  - Os argumentos não são indicados, mesmo que existam

```
float sum();
void main() { .... }
float sum( float a, float b ) {...}
```

Funções de tipo não inteiro

- Protótipo
  - Inclui também a quantidade e tipos dos parâmetros

```
float sum(int  , int);
void main() { .... }
float sum( float a, float b ) {...}
```

  - E no caso de funções sem argumento?
    - Declarar lista como void

Lista de argumentos variável

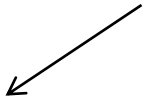
- Uma função pode admitir uma lista com tamanho e tipos variáveis
- Exemplo?

Lista de argumentos variável

- Uma função pode admitir uma lista com tamanho e tipos variáveis
- Exemplo?
  - printf(char *string, ...)
- Utilização de uma macro
  - <stdarg.h>

Exemplo

INDICADOR DE LISTA VARIÁVEL DE PARÂMETROS



```
#include <stdarg.h>

double average( int i, ... )
{
    double total = 0;
    int j;

    va_list ap;          //Cria objeto de manipulação
    va_start( ap, i ); //Inicializa o objeto ap

    for ( j = 1; j <= i; j++ )
        total += va_arg( ap, double )

    va_end(ap);           //Limpando a memória

    return total / i;
}
```

Lista de argumentos variáveis

- va_start(ap, i)
  - ap é o nome da estrutura a ser inicializada
  - o segundo argumento é o nome da última variável antes da elipse (...)
- va_arg(ap, double)
  - Cada chamada retorna o valor passado
  - O segundo argumento representa o tipo do dado esperado na chamada
- va_end(ap): limpeza da estrutura criada

Exercício

- Usando a biblioteca stdarg.h, implemente a função com o seguinte protótipo
  - void printf(char *str, ...);
- Considere como formatação possível apenas %u
  - Range do %u: 0 até 4294967295

Enumeração

- Conjunto de constantes

```
enum months { JAN, FEV, MAR, ABR, MAI, JUN, JUL,
    AGO, SET, OUT, NOV, DEZ };
```

- Primeiro valor é zero, se nenhum valor é especificado
- Outros valores são incrementados de 1

```
enum months { JAN = 1, FEV, MAR, ABR, MAI, JUN, JUL,
    AGO, SET, OUT, NOV, DEZ };
```

Enumeração

```
int main( void )
{
   enum months month; // can contain any of the 12 months

   // initialize array of pointers
   const char *monthName[] = { "", "January", "February",
   "March", "April", "May", "June", "July", "August",
   "September", "October", "November", "December" };

   // loop through months
   for ( month = JAN; month <= DEC; ++month ) {
      printf( "%2d%11s\n", month, monthName[ month ] );
   } // end for
} // end main
```

Arquivos

- Sequência de bytes que reside no disco e não na memória principal
- Endereçamento
  - Sequencial
- Manipulação
  - FILE *fd; //Biblioteca stdio.h

Tipos de arquivos em C

- Arquivos de textos
  - Facilmente editável em programas de edição de texto
- Arquivos binários

Arquivos de texto

- São gravados como caracteres de 8 bits
- A escrita é feita da mesma forma que os dados seriam impressos na tela
- Conversão de dados não texto para dados tipo texto

Quando usar arquivos de texto?

- Exemplo
  - int x = 38472039;
  - char str[8] = “38472039”

- Na memória x ocupa 32 bits
- No arquivo texto, ocupará 8*8 = 64 bits
- A escolha de arquivo texto depende da aplicação

Arquivo binário

- Os dados são gravados exatamente como são armazenados na memória

- Como não há conversão
  - A leitura e escrita são mais rápidas
  - Os arquivos são menores

Manipulação de arquivos de texto

- FILE *fd = fopen(arqName, “r”);
  - Abre para leitura
- FILE *fd = fopen(arqNAme, “w”);
  - Abre para escrita

- Depois de usar o arquivo, é necessário fechá-lo
  - fclose(fd);

Manipulação de arquivos binários

- FILE *fd = fopen(arqName, “rb”);
  - Abre para leitura
- FILE *fd = fopen(arqNAme, “wb”);
  - Abre para escrita

- Depois de usar o arquivo, é necessário fechá-lo
  - fclose(fd);

Leitura - fscanf

- fscanf(FILE *fd, char *str, …)
  - fd = descritor do arquivo a ser lido
  - str = formato a ser lido
  - ... = lista de variáveis a serem lidas

- Se fd == stdin, fscanf equivale a scanf

Escrita - fprintf

- fprintf(FILE *fd, char *str, …)
  - fd = descritor do arquivo a ser escrito
  - str = formato a ser escrito
  - ... = lista de variáveis a serem escritas

- Se fd == stdout, fprintf equivale a printf

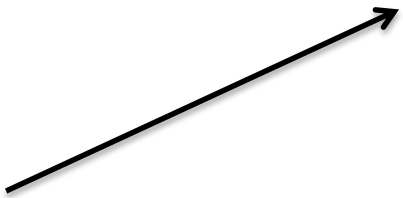
```c
int main( void) {

    int x, n = 0, k;
    double soma = 0;

    FILE *entrada = fopen( "dados.txt", "r");
    if (entrada == NULL)
        exit( EXIT_FAILURE);

    while ( 1 ) {
        k = fscanf( entrada, "%d", &x);
        if (k != 1) break;
        soma += x;
        n += 1;
    }
    fclose( entrada);
    printf( "A média dos números é %f\n", soma / n);
    return EXIT_SUCCESS;
}
```

dados.txt
1 3 6 1 2 93 12



putc e getc

- int putc (int character, FILE * stream);
  - Escreve caracter no arquivo
  - Retorna o caracter escrito caso tenha sucesso, cc. retorna EOF
- int getc (FILE * stream);
  - Retorna caracter lido
  - Caso erro, retorna EOF

Escrita em arquivos binários

- Mais adequada para a escrita de dados mais complexos, como structs

```
int fwrite (void *buffer, int bytes,
            int count, FILE *fp)
```

Retorna número de unidades escritas com sucesso

fwrite

int fwrite (void *buffer, int bytes,
int count, FILE *fp)

- **buffer**: ponteiro genérico para os dados
- **bytes**: tamanho, em bytes, de cada unidade de dado a ser gravada
- **count**: total de unidades a gravar
- **fp**: ponteiro para o arquivo

fwrite

```c
int main() {

    //Abre para escrita
    FILE *f = fopen ( "file.txt", "wb");
    if ( f == NULL ) exit(1);

    //Vetor de inteiros
    int total_gravado, v[5] = {0,1,2,3,4};

    //Escreve o vetor
    total_gravado = fwrite(v, sizeof(int), 5, f);
    fclose(f);

    return 0;

}
```

fread

- Leitura de bloco de dados de um arquivo
- Usado em conjunto com fwrite

```
int fread ( void *buffer, int bytes,
            int count, FILE *fp );
```

Retorna número de unidades lidas com sucesso

```c
int main() {

    //Abre para escrita
    FILE *f = fopen ( "file.txt", "rb");
    if ( f == NULL ) exit(1);

    //Vetor de inteiros
    int total_lido, v[5];// = {0,1,2,3,4};

    //Escreve o vetor
    total_lido = fread(v, sizeof(int), 5, f);
    fclose(f);

    if (total_lido != 5) exit(1);

    printf( "%d %d %d %d %d\n", v[0], v[1], v[2], v[3], v[4]);

    return 0;
}
```

Acesso randômico em arquivos

- Em geral o acesso é feito de modo sequencial
- A linguagem C fornece ferramentas para realizar leitura e escrita randômica

```
int fseek(FILE *fp, long numbytes, int origem)
```

- numbytes pode ser negativo

Acesso randômico

- A origem pode assumir as seguintes constantes (definida em *stdio.h*)

SEEK_SET (0): início do arquivo

SEEK_CUR (1): posição atual do arquivo

SEEK_END (2): final do arquivo

Acesso randômico em arquivos

```c
struct cadastro { char name[30], int age };

int main () {

    //Abre para escrita
    FILE *f = fopen ( "file.txt", "wb");
    if ( f == NULL ) exit(1);

    //Inicializando vetor de cadastro
    struct cadastro cad[4] = { "a", 1, "b", 2, "c", 3, "d", "4" };

    //Escreve no arquivo
    fwrite( cad, sizeof( struct cadastro ), 4, f );

    return 0;
}
```

Acesso randômico em arquivos

```
//Abre para escrita
FILE *f = fopen ( "file.txt", "rb");
if ( f == NULL ) exit(1);

//Inicializando vetor de cadastro
struct cadastro c;

//Acesso nao sequencial
fseek( f, 2 * sizeof( struct cadastro ), SEEK_SET );
fread( &c, sizeof( struct cadastro ), 1, f );
fclose(f);

//Acessa o registro com dados name == "c" e age == 3
printf( "%s %d\n", c.name, c.age );

return 0;
```

Acesso randômico

- void rewind (FILE *fd)
  - Retorna ao início do arquivo
  - Evita abrir e fechar o arquivo para ir ao início

Acesso randômico

```c
struct cadastro { char name[30], int age };

int main() {

    //Abre para escrita
    FILE *f = fopen ( "file.txt", "rb");
    if ( f == NULL ) exit(1);

    //Inicializando vetor de cadastro
    struct cadastro c;

    //Acesso nao sequencial
    fseek( f, 2 * sizeof( struct cadastro ), SEEK_SET );
    rewind(f);
    fread( &c, sizeof( struct cadastro ), 1, f );
    fclose(f);

    //Acessa o registro com dados name == "a" e age == 1
    printf( "%s %d\n", c.name, c.age );

    return 0;
}
```