Estruturas Condicionais e de Repetição

Comando while

- Deseja-se calcular o valor de: 1 + 2 + 3 + ... + N.

- Observação: não sabemos, *a priori*, quantos termos serão somados, pois o valor de N é estabelecido dinamicamente.

- Para se calcular esta soma, o programa p09.c utiliza o comando while.
  - O comando while permite que um conjunto de instruções seja executado tantas vezes quantas forem necessárias, enquanto uma condição for verdadeira.

Comando while

- Quando um programa executa um conjunto de instruções repetidas vezes, diz-se que o programa está realizando um processamento iterativo.

- Cada execução do conjunto de instruções denomina-se uma iteração. Exemplo de uso do comando while:

```
s = 0;
i = 1;
while (i <= 3)
{
    s = s + i;
    i++;
}
```

Instante	s	i	(i <= 3)
inicial	0	1	V
1ª Iteração	0 + 1 = 1	1 + 1 = 2	V
2ª Iteração	1 + 2 = 3	2 + 1 = 3	V
3ª Iteração	3 + 3 = 6	3 + 1 = 4	F

Comando while

- A execução do programa p09.c sempre irá terminar, pois i será maior do que N em algum momento.

- Porém, pode a execução de um programa com processamento iterativo não terminar? Observe:

```
s = 0;
i = 0;
while (i < 3)
{
    i--;
    s = s + i;
}
```

Este laço ou loop nunca irá terminar! (Erro de lógica)

```
s = 0;
i = 0;
while (i > -3)
{
    i--;
    s = s + i;
}
```

Maneira correta

Problema 9

- Dado o valor da variável N, determine a soma dos números inteiros de 1 a N.

```c
#include <stdio.h>
#include <stdlib.h>

int main(int argc, char *argv[])
{
  int N;
  int i, s;

  printf("Digite o valor de N: ");
  scanf("%d", &N);
  s = 0;
  i = 1;
  while (i <= N)
  {
    s = s + i;
    i++;
  }
  printf("Soma = %d\n", s);
  system("PAUSE");
  return 0;
}
```

Comando while

- Atenção!

  Em alguns casos, o loop infinito pode ser desejável. Exemplo: um programa que monitora um reator nuclear deve estar sempre em execução.

- Neste caso, pode-se escrever:

```
while (1)
{
    ...
}
```

Comando do-while

- Outra forma de repetir um conjunto de instruções é com o comando do-while.

```
s = 0;
i = 1;
do
{
    s = s + i;
    i++;
}
while (i <= N);
```

Comando do-while

```
s = 0;
i = 1;
while (i <= N)
{
    s = s + i;
    i++;
}
```

Comando while

- Veja que no comando while, a condição é testada antes da execução das instruções, ao contrário do comando do-while. O que acontece para N=0?

Problema 10

- Suponha que soma (+) e subtração (-) são as únicas operações disponíveis em C. Dados dois números inteiros positivos A e B, determine o quociente e o resto da divisão de A por B.

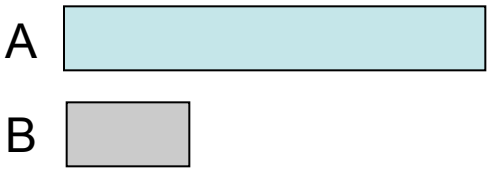
Algoritmos estruturados

- Para resolver o problema 10, precisamos de um algoritmo:

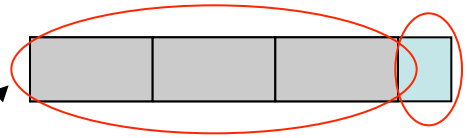
> Seqüência finita de instruções que, ao ser executada, chega a uma solução de um problema.

- Para escrever este algoritmo, podemos usar a seguinte idéia:

1. Representar os números A e B por retângulos de larguras proporcionais aos seus valores;
2. Verificar quantas vezes B cabe em A.



A = 7, B = 2



Quociente de A/B: nº de vezes que B cabe em A.

O que sobra em A é o resto da divisão.

Algoritmos estruturados

- Pode-se escrever este algoritmo como:

1. Sejam A e B os valores dados;
2. Atribuir o valor 0 ao quociente (q);
3. Enquanto B couber em A:

{

4. Somar 1 ao valor de q;
5. Subtrair B do valor de A;

}

9. Atribuir o valor final de A ao resto (r).

Problema 10

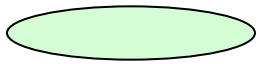
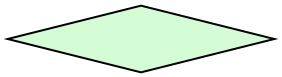
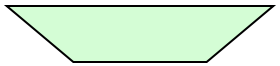
- Suponha que soma (+) e subtração (-) são as únicas operações disponíveis em C. Dados dois números inteiros positivos A e B, determine o quociente e o resto da divisão de A por B.

```c
#include <stdio.h>
#include <stdlib.h>

int main(int args, char * arg[])
{
  int A, B;
  int q, r;

  printf("Digite os valores de A e B: ");
  scanf("%d %d", &A, &B);
  q = 0;
  while (B <= A)
  {
    q++;
    A = A - B;
  }
  r = A;
  printf("Quociente = %d e Resto = %d\n", q,r);
  system("PAUSE");
  return 0;
}
```

Fluxograma

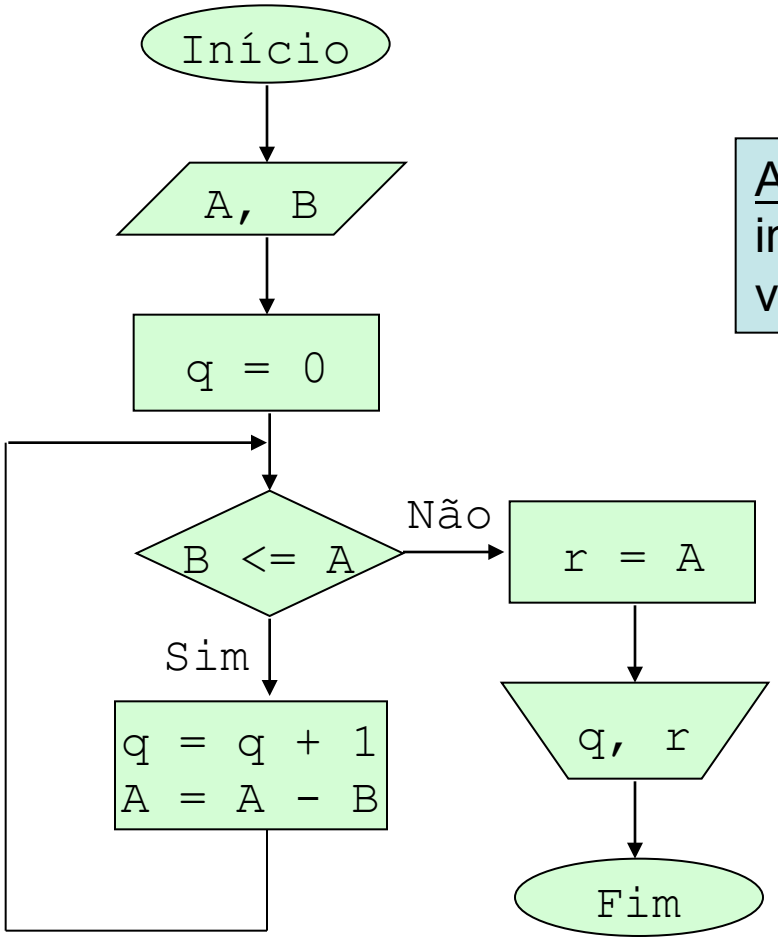
- É conveniente representar algoritmos por meio de fluxogramas (diagrama de blocos).

- Em um fluxograma, as operações possíveis são representadas por meio de figuras:

Figura	Usada para representar
	Início ou fim.
	Atribuição.
	Condição.
	Leitura de dados.
	Apresentação de resultados.
	Conexão entre partes do algoritmo.
	Fluxo de execução.

Fluxograma

- Exemplo: o algoritmo para o Problema 10 pode ser representado pelo seguinte fluxograma.



Atenção: observe que um algoritmo não inclui detalhes, tais como declaração de variáveis, mensagens a serem exibidas, etc.

Programação Estruturada

- Um algoritmo tem sempre um único bloco início e deve conter, pelo menos, um bloco fim. A execução de um algoritmo deve ser feita seguindo-se as setas.

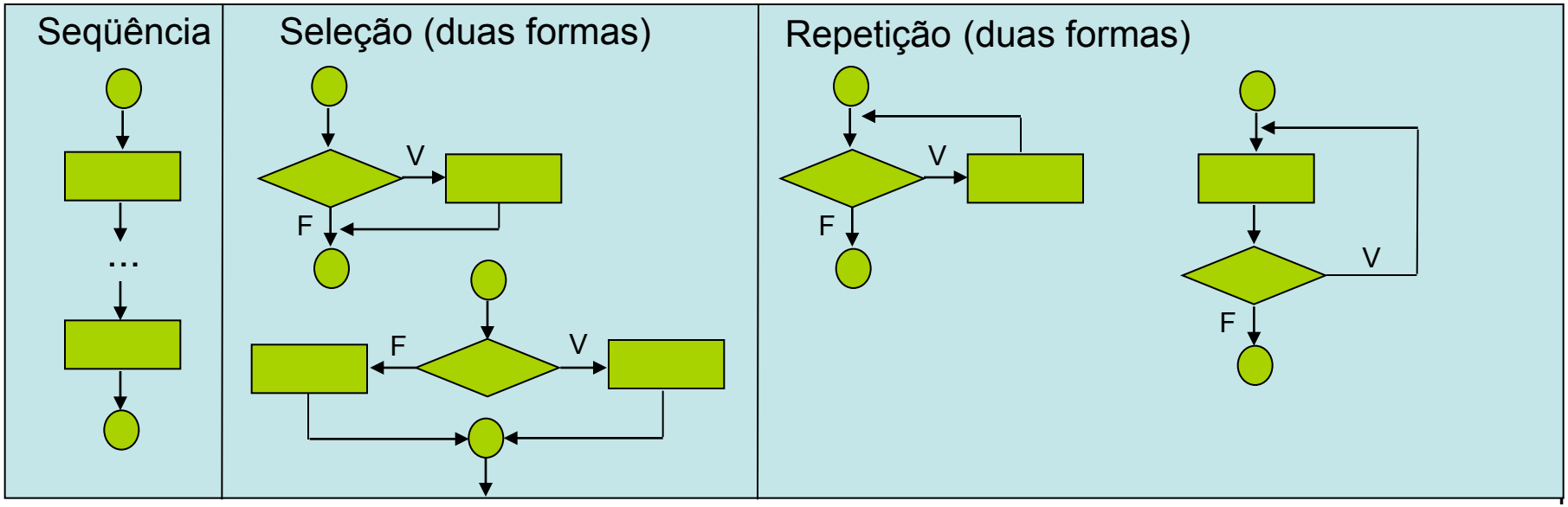
- Em geral, a construção de um algoritmo é uma tarefa mais difícil do que codificar o algoritmo em uma linguagem de programação.

- Porém, a construção de algoritmos pode se beneficiar de técnicas, como a Programação Estruturada.

Programação Estruturada

- Na Programação Estruturada, os programas são construídos usando-se apenas três estruturas: seqüência, seleção e repetição.

- Cada estrutura tem apenas um único ponto de entrada e um único ponto de saída (círculos).



Programação Estruturada

- Atenção!
  - Nestas estruturas, os retângulos são utilizados para indicar qualquer ação, incluindo leitura de dados ou exibição de resultados.

- Construir programas estruturados corresponde a combinar estas estruturas de duas maneiras:
  - Regra do empilhamento: o ponto de saída de uma estrutura pode ser conectado ao ponto de entrada de outra estrutura.
  - Regra do aninhamento: um retângulo de uma estrutura pode ser substituído por uma outra estrutura qualquer.

Programação Estruturada

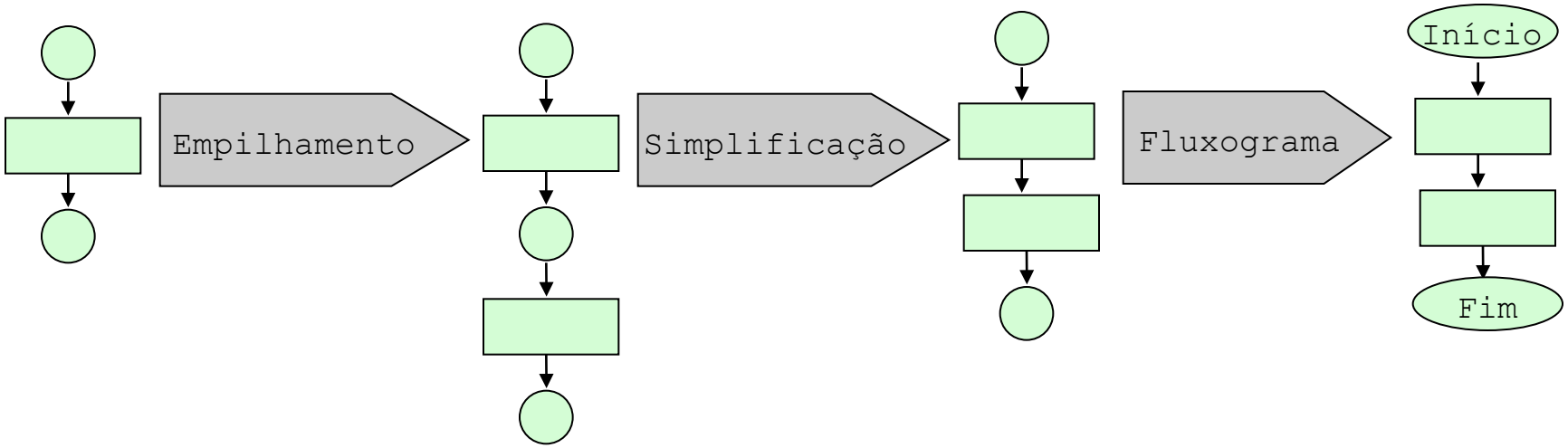
- Estas regras podem ser aplicadas quantas vezes forem necessárias e em qualquer ordem.

- Na construção de fluxogramas, pode-se substituir o primeiro ponto de entrada e os últimos pontos de saída por ovais (início e fim).

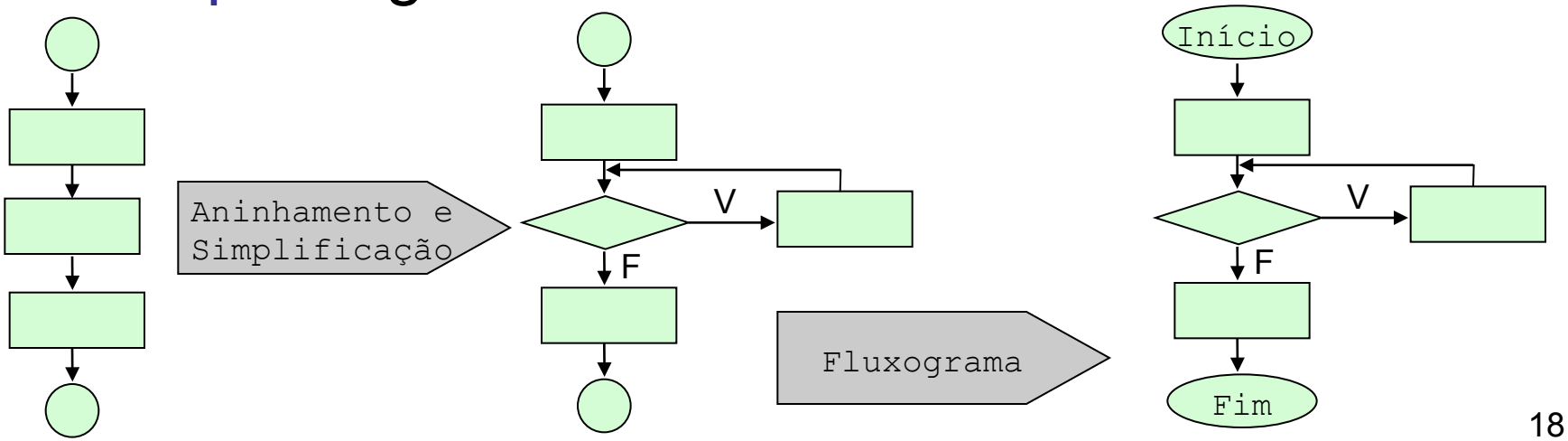
- Os demais pontos de entrada e saída podem ser removidos.

Programação Estruturada

- Exemplo: aplicação da regra de empilhamento.



- Exemplo: algoritmo do Problema 10.



Problema 11

- Dados dois números inteiros A e B, determinar o máximo divisor comum destes dois números.

Problema 11

- Como calcular o máximo divisor comum entre dois números A e B, representado por mdc(A,B)?

- Método das divisões sucessivas: efetua-se várias divisões até chegar em uma divisão exata. Suponha que se deseje calcular mdc(48,30).

1. Divide-se o nº maior pelo nº menor: 48/30 = 1 (resto 18).
2. Divide-se o divisor anterior pelo resto anterior e, assim sucessivamente:

30/18 = 1 (resto 12)

18/12 = 1 (resto 6)

12/6 = 2 (resto 0 – divisão exata)

mdc(48,30) = 6, que corresponde ao divisor da divisão exata.

Problema 11 - Algoritmo

- Um algoritmo para este problema pode ser escrito como:

1. Enquanto B for diferente de zero:
{
    2. r = resto da divisão de A por B;
    3. A = B;
    4. B = r;
}
5. mdc = A;

Problema 11

- Dados dois números inteiros A e B, determinar o máximo divisor comum destes dois números.

```c
// Programa p11.c
#include <stdio.h>
#include <stdlib.h>

int MDC(int A, int B)
{
  int r;

  while (B != 0)
  {
    r = A % B;
    A = B;
    B = r;
  }
  return A;
}

int main(int args, char * arg[])
{
```

Problema 11 - Programa principal

- O programa principal aparece como um conjunto de instruções do comando do-while:

```
int main(int args, char * arg[])
{
  int A, B, m;
  char c;

  do
  {
    printf("Digite os valores de A e B: ");
    scanf("%d %d", &A, &B);
    m = MDC(A,B);
    printf("MDC(%d,%d) = %d\n",A,B,m);
    printf("Continua? (S/N) ");
    do
    {
      c = getche();
      if ((c != 'S') && (c != 'N'))
      {
        putch('\a');
        putch('\b');
      }
    }
    while ((c != 'S') && (c != 'N'));
    printf("\n");
  }
  while ((c == 'S') || (c == 's'));
  system("PAUSE");
  return 0;
}
```

Após exibir o valor do mdc, o programa exibe a mensagem:

Continua? (S/N)

Espera-se que o usuário digite S ou N, caracteres que serão lidos pela função getche.

A função getche retorna o código ASCII do caractere lido.

O loop será executado enquanto c for igual a ‘S’ ou a ‘s’.

Função toupper

- Para evitar a comparação com letras maiúsculas e minúsculas, pode-se usar a função toupper:

```
do
{
  printf("Digite os valores de A e B: ");
  scanf("%d %d", &A, &B);
  m = MDC(A,B);
  printf("MDC(%d,%d) = %d\n",A,B,m);
  printf("Continua? (S/N) ");
  do
  {
    c = toupper(getche());
    if ((c != 'S') && (c != 'N'))
    {
      putch('\a');
      putch('\b');
    }
  }
  while ((c != 'S') && (c != 'N'));
  printf("\n");
}
while ((c == 'S') || (c == 's'));
```

Verifica se o valor de seu parâmetro corresponde ao código ASCII de uma letra minúscula.

Caso afirmativo, retorna o código da letra maiúscula correspondente.

Caso negativo, retorna o próprio valor do parâmetro.

Comandos if-else interrelacionados

- O programa p12.c apresenta diversos comandos if-else interrelacionados.

- Exemplo: imagine uma função que recebe como parâmetro um inteiro representando o número de um mês e retorna o número de dias deste mês (considere que fevereiro tem sempre 28 dias).

Comandos if-else interrelacionados

```
int dias_do_mes(int mes)
{
  int dias;
  if (mes == 1)
    dias = 31;
  else if (mes == 2)
    dias = 28;
  else if (mes == 3)
    dias = 31;
  else if (mes == 4)
    dias = 30;
  else if (mes == 5)
    dias = 31;
  else if (mes == 6)
    dias = 30;
  else if (mes == 7)
    dias = 31;
  else if (mes == 8)
    dias = 31;
  else if (mes == 9)
    dias = 30;
  else if (mes == 10)
    dias = 31;
  else if (mes == 11)
    dias = 30;
  else if (mes == 12)
    dias = 31;
  else
    dias = 0;
  return dias;
}
```

Comandos if-else interrelacionados

- Uma outra forma de escrever esta função, mas ainda com comandos if-else interrelacionados é:

```
int dias_do_mes(int mes)
{
  int dias;
  if ((mes == 1) || (mes == 3) || (mes == 5) ||
      (mes == 7) || (mes == 8) || (mes == 10) ||
      (mes == 12))
    dias = 31;
  else if (mes == 2)
    dias = 28;
  else if ((mes == 4) || (mes == 6) || (mes == 9) ||
           (mes == 11))
    dias = 30;
  else
    dias = 0;
  return dias;
}
```

Comando switch

- A demanda por comandos if-else interrelacionados é muito comum em programação.

- Assim, a linguagem C disponibiliza um comando especial para tais situações: switch. A sintaxe deste comando é a seguinte:

```
switch (expressão)
{
   case constante-1:
      comandos-1;
   case constante-2:
      comandos-2;
   ...
   default:
      comandos-n;
}
```

Comando switch

- Com o comando switch, a função dias_do_mes pode ser reescrita como:

```
int dias_do_mes(int mes)
{
  int dias;
  switch (mes)
  {
    case 1:
    case 3:
    case 5:
    case 7:
    case 8:
    case 10:
    case 12:
      dias = 31;
      break;
    case 2:
      dias = 28;
      break;
    case 4:
    case 6:
    case 9:
    case 11:
      dias = 30;
      break;
    default:
      dias = 0;
  }
  return dias;
}
```

Comando switch

- Este comando permite que, de acordo com o valor de uma expressão, seja executado um ou mais comandos dentre uma série de alternativas.

- O caso cuja constante for igual ao valor da expressão será selecionado para execução.

- Atenção!
  - Os comandos associados a este caso e todos os comandos seguintes serão executados em sequência até o final do comando switch.
  - Para evitar a execução de todos os comandos seguintes, usa-se o comando break.

Comando switch

- Como assim?

Para mes = 2:

- Com o uso de break:

```
dias = 28;
```

Para mes = 2:

- Sem o uso de break:

```
dias = 28;
dias = 30;
dias = 0;
```

```
int dias_do_mes(int mes)
{
  int dias;
  switch (mes)
  {
    case 1:
    case 3:
    case 5:
    case 7:
    case 8:
    case 10:
    case 12:
      dias = 31;
      break;
    case 2:
      dias = 28;
      break;
    case 4:
    case 6:
    case 9:
    case 11:
      dias = 30;
      break;
    default:
      dias = 0;
  }
  return dias;
}
```

Comandos break e continue

- Em alguns programas, durante um processamento iterativo, pode ser necessário:
  - Encerrar o processamento iterativo independentemente do valor da condição do laço;
  - Executar apenas parcialmente uma iteração, ou seja, executar somente algumas das instruções do laço da repetição.
- Para encerrar um processamento iterativo, independentemente do valor da condição do laço, deve-se usar o comando break.

Comandos break e continue

- Exemplo: dados os valores N (int) e A (float), determine a partir de qual termo o valor de:

$$s = 1 + \frac{1}{2} + \frac{1}{3} + \ldots + \frac{1}{N}$$

é maior do que A.

Suponha N = 10 e A = 2.

Instante	Valor de s
1º termo	1.000000
2º termo	1.500000
3º termo	1.833333
4º termo	2.083333

A partir do quarto termo s > A.

Comandos break e continue

- Um programa para resolver este problema pode ser escrito como:

```
i = 1;
s = 0;
while (i <= N)
{
  s = s + 1.0/i;
  if (s > A)
  {
    printf("Numero de termos = %d\n", i);
    break;
  }
  i++;
}
```

- Neste caso, a condição do laço controla apenas o número de termos do somatório.
- O laço pode ser encerrado quando s > A, usando-se o comando break.

Comandos break e continue

- Para executar somente algumas das instruções do laço, mas sem encerrar a repetição: comando continue.
- Exemplo: ler a idade e o peso de N pessoas e determinar a soma dos pesos das pessoas com mais de 30 anos.

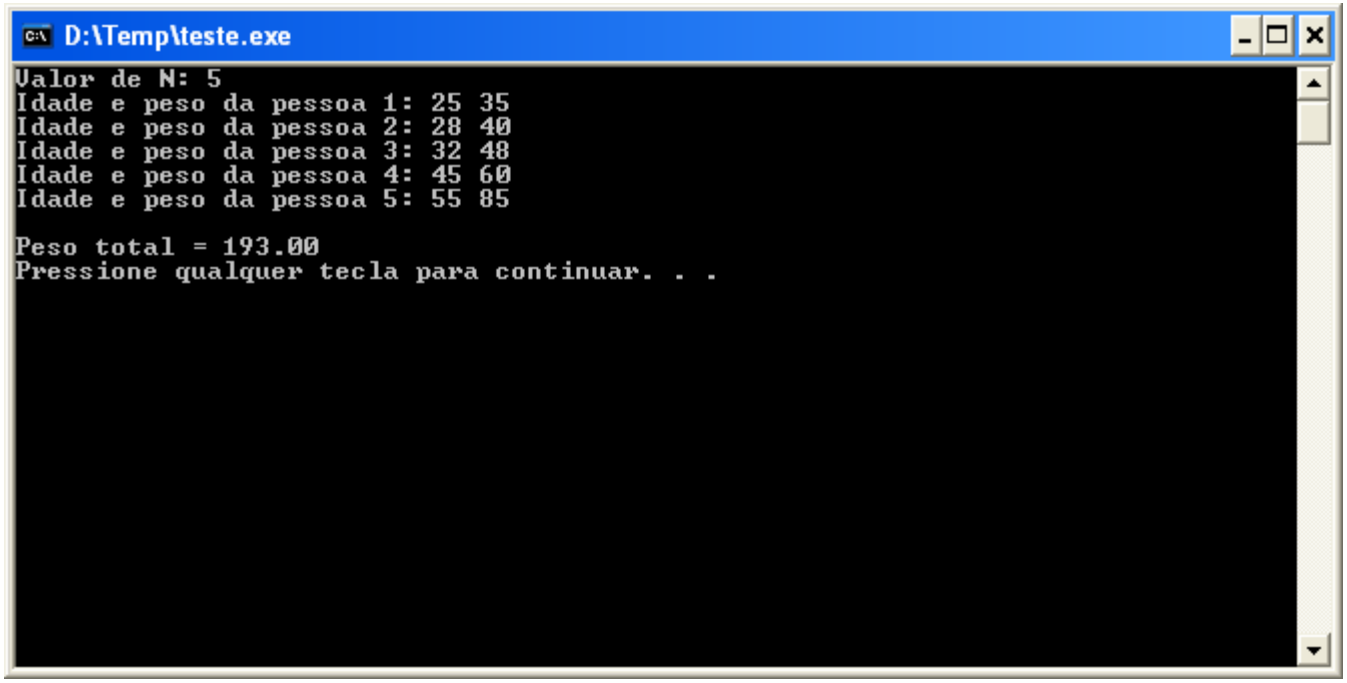
```
printf("Valor de N: ");
scanf("%d", &N);
i = 0;
s = 0;
while (i < N)
{
   i++;
   printf("Idade e peso da pessoa %d: ",i);
   scanf("%d %f",&idade,&peso);
   if (idade <= 30)
      continue;
   s = s + peso;
}
printf("Peso total = %.2f\n",s);
```

O comando continue faz com que a instrução s = s + peso não seja executada quando idade <= 30.

Ou seja, ele volta a execução para o início do laço.

Comandos break e continue

- Resultado da execução:



- Peso total = ~~35~~ + ~~40~~ + 48 + 60 + 85 = 193

Comando for

- O comando for é similar ao comando while, porém, em alguns casos, seu uso é mais prático
- Sua sintaxe é
  - for (<inicialização>; <condição>; <incremento>)

- Exemplo: escreva um programa para calcular o fatorial de um inteiro n usando o comando while e o comando for.