Universidade de São Paulo – São Carlos
Instituto de Ciências Matemáticas e de Computação

# Depuração e teste de programas C

Profa Rosana Braga
(adaptado de material do
prof. André Takeshi Endo)

1º semestre de 2010

# Roteiro

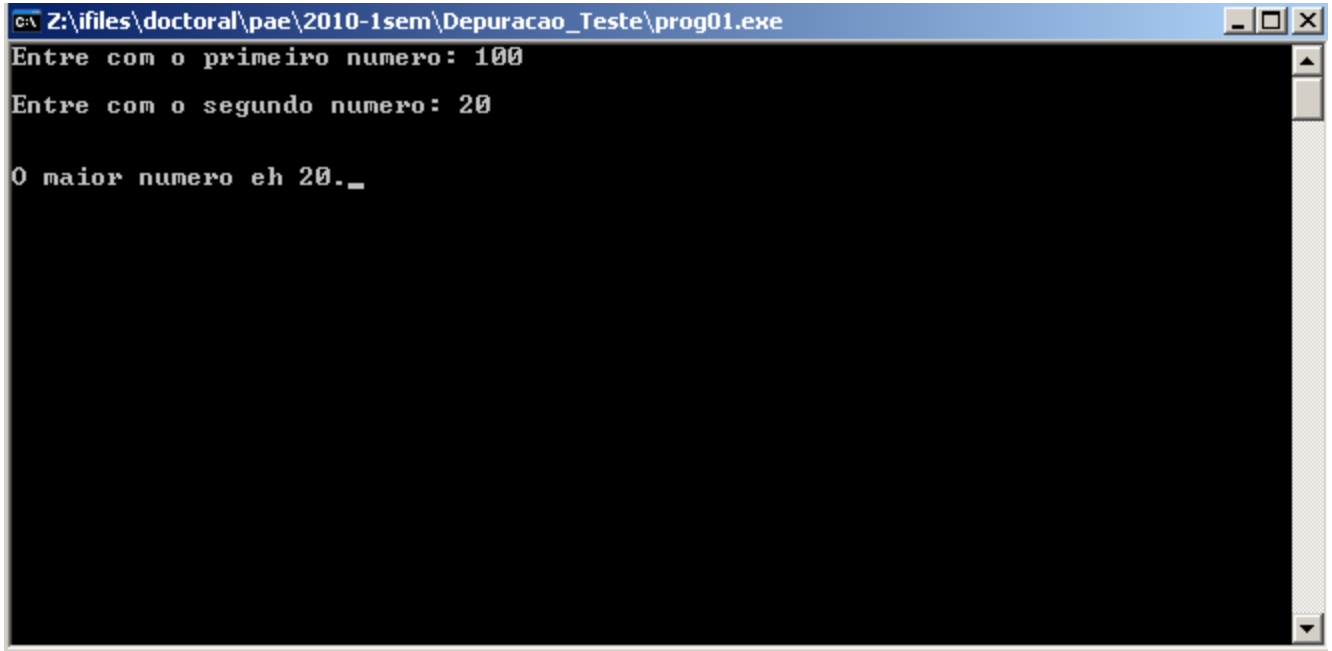
- Teste de Software
  - Definições
  - Caso de teste
- Depuração de programas
  - Dev-C++
  - Gdb + gcc (Linux)

# Teste de Software: definições

- Executar um programa com a intenção de encontrar defeitos.
- Defeito / Bug / Erro
  - Instrução ou comando incorreto
- Falha
  - Produção de uma saída incorreta

# Teste de Software: definições

- Exemplo: Leia dois inteiros e imprima o maior.



# Teste de Software: definições

- Exemplo: Leia dois inteiros e imprima o maior.

# Teste de Software: definições

- Exemplo: Leia dois inteiros e imprima o maior.

```
#include<stdio.h>
#include<conio.h>

int main(void)
{
   int num1, num2;

   printf("Entre com o primeiro numero: ");
   scanf("%d", &num1);
   printf("\nEntre com o segundo numero: ");
   scanf("%d", &num2);

   if(num1 < num2)
      printf("\n\nO maior numero eh %d.", num1);
   else
      printf("\n\nO maior numero eh %d.", num2);

   getch();
}
```

# Teste de Software: definições

- Exemplo: Leia dois inteiros e imprima o maior.

```
#include<stdio.h>
#include<conio.h>

int main(void)
{
   int num1, num2;

   printf("Entre com o primeiro num
   scanf("%d", &num1);
   printf("\nEntre com o s                numero:
   scanf("%d", &num

   if(num1 < num2)
      printf("\n\nO maior numero eh %d.", num1);
   else
      printf("\n\nO maior numero eh %d.", num2);

   getch();
}
```

O **defeito** deste programa é a troca do sinal. O correto seria: if(num1 > num2)

# Teste de software: definições

- Caso de teste
  - <entradas; saídas esperadas>

- Exemplo: Faça um programa que receba como entrada um inteiro positivo e responda se o número é primo ou não.
  - < 5; primo >
  - < 49; nao eh primo >
  - < 37; primo >
  - < 10; nao eh primo >
  - < -15; entrada invalida >
  - < icc; entrada invalida >

# Depuração: Conceitos

- *Debugging*
- Processo de encontrar e corrigir defeitos
- Como encontrar o defeito?
  - Mensagens do compilador
  - Revisão de código
  - Ferramentas de depuração

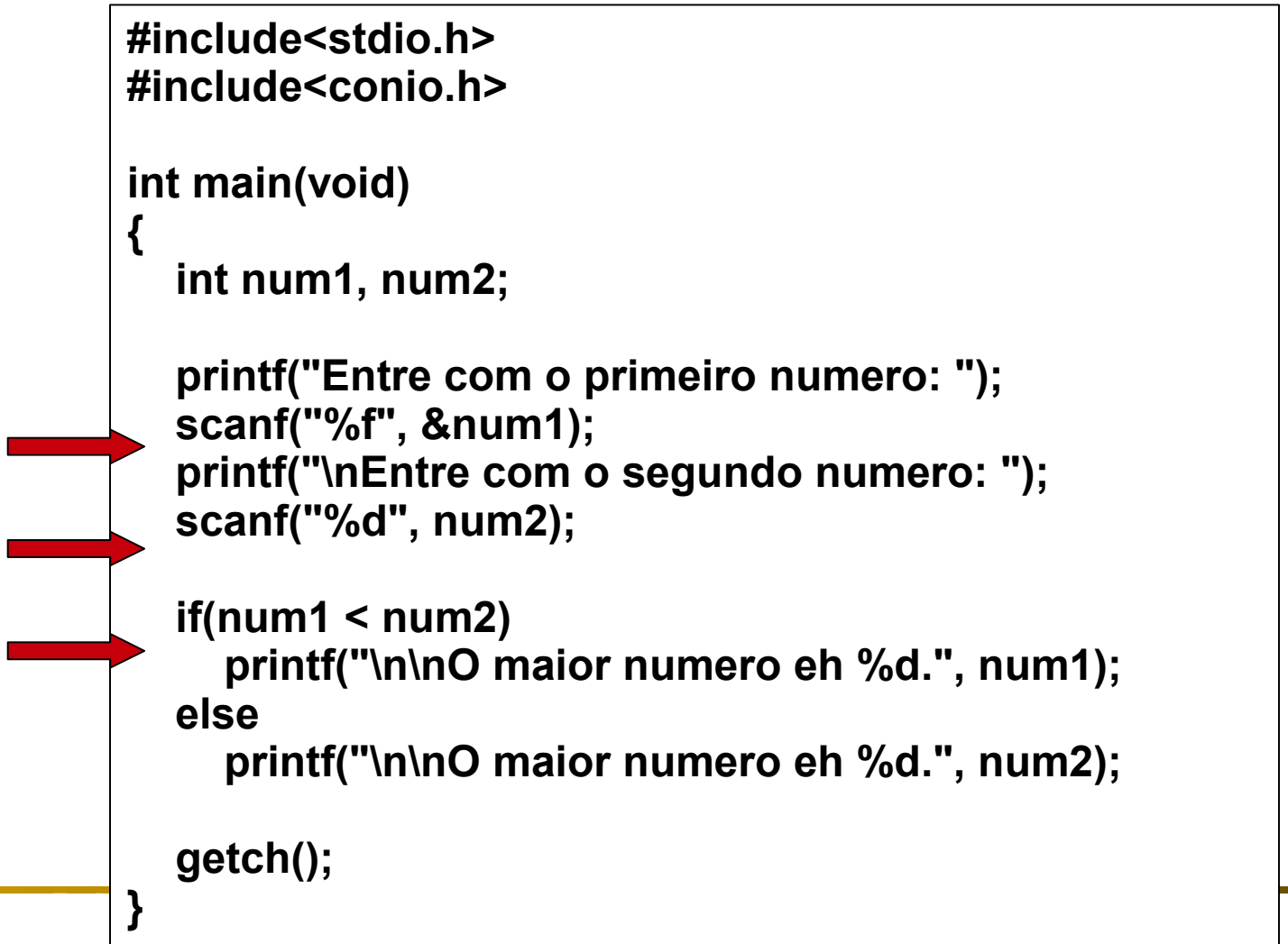
# Depuração: Revisão do código

- 3 defeitos:

```
#include<stdio.h>
#include<conio.h>

int main(void)
{
   int num1, num2;

   printf("Entre com o primeiro numero: ");
   scanf("%f", &num1);
   printf("\nEntre com o segundo numero: ");
   scanf("%d", num2);

   if(num1 < num2)
      printf("\n\nO maior numero eh %d.", num1);
   else
      printf("\n\nO maior numero eh %d.", num2);

   getch();
}
```

# Depuração: Revisão do código

- 3 defeitos:

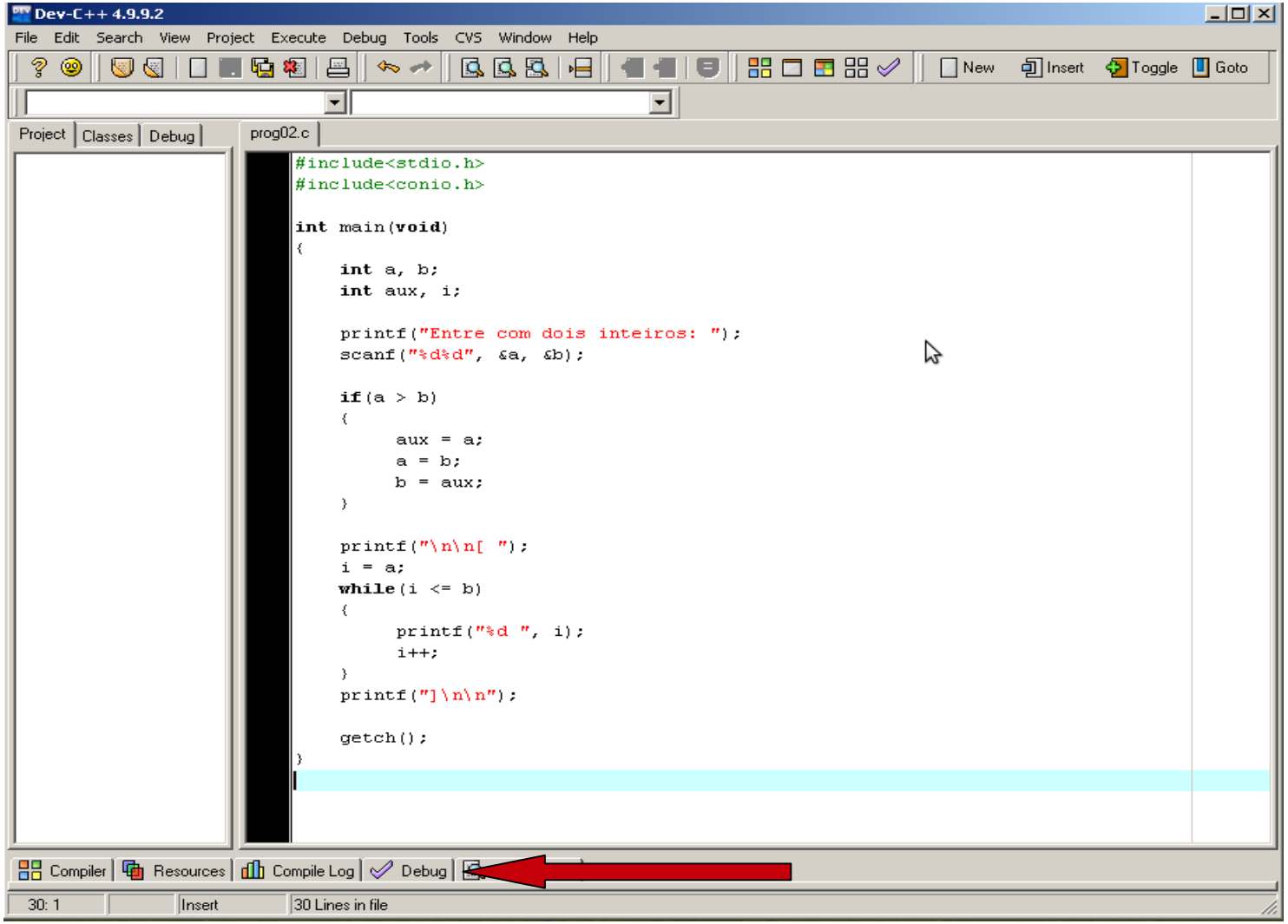


```
#include<stdio.h>
#include<conio.h>

int main(void)
{
   int num1, num2;

   printf("Entre com o primeiro numero: ");
   scanf("%f", &num1);
   printf("\nEntre com o segundo numero: ");
   scanf("%d", num2);

   if(num1 < num2)
      printf("\n\nO maior numero eh %d.", num1);
   else
      printf("\n\nO maior numero eh %d.", num2);

   getch();
}
```

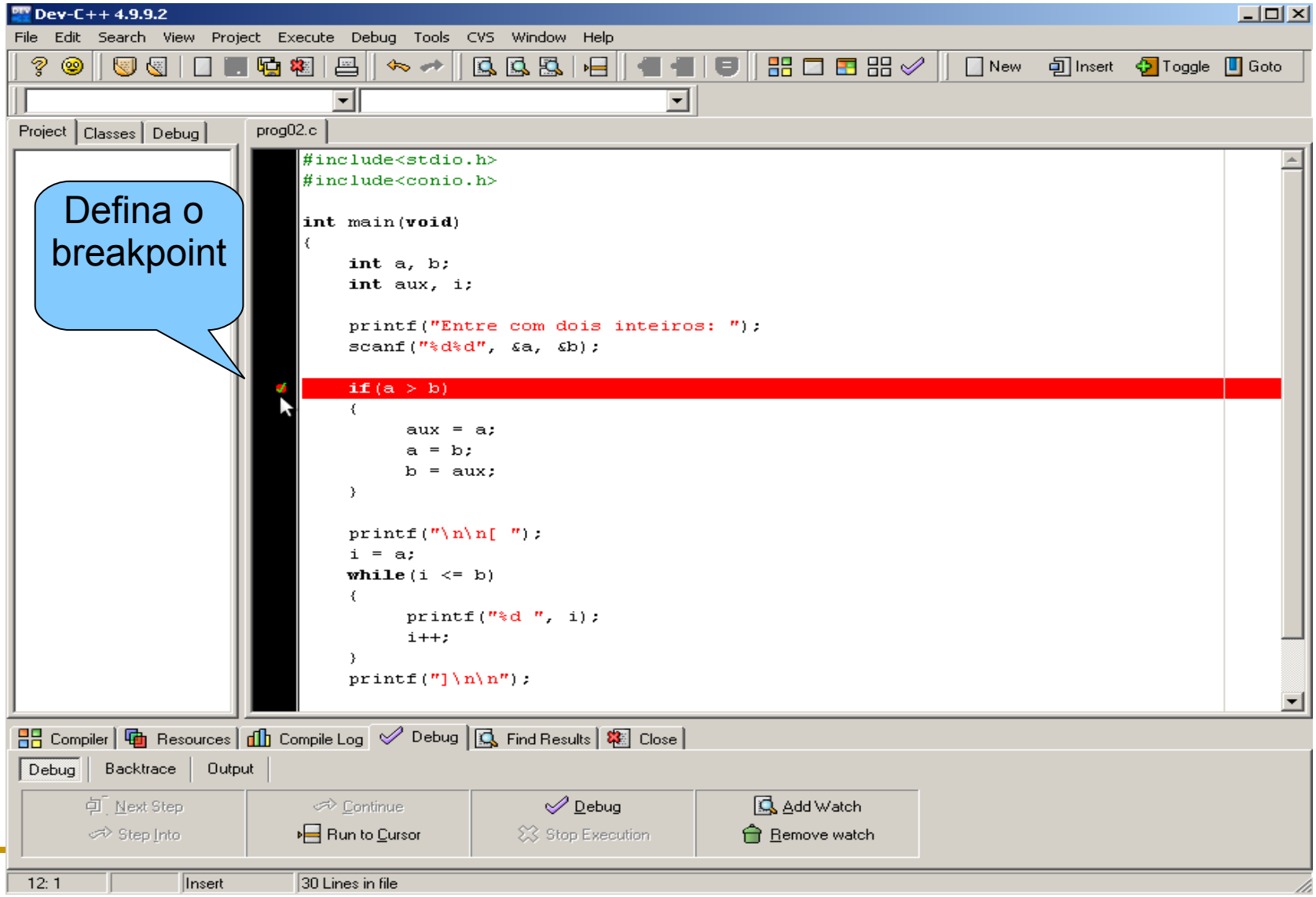
# Depuração: Dev-C++

- gdb - GNU Project Debugger <link>
- Recursos gráficos
- Breakpoints
  - Até que ponto o programa deve executar normalmente
- Watch
  - Quais variáveis devem ser acompanhadas?

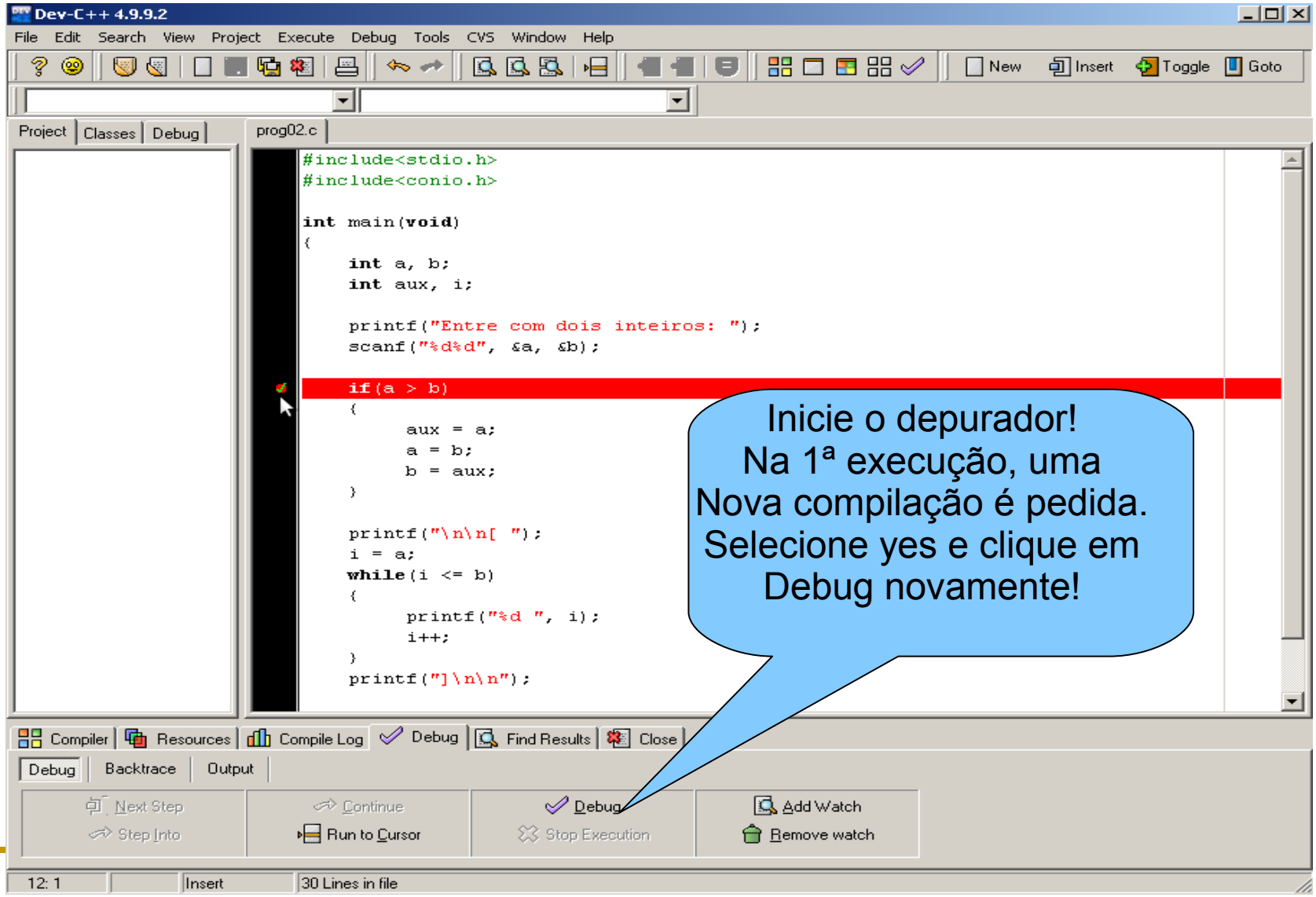
# Depuração: Dev-C++(passo 1)



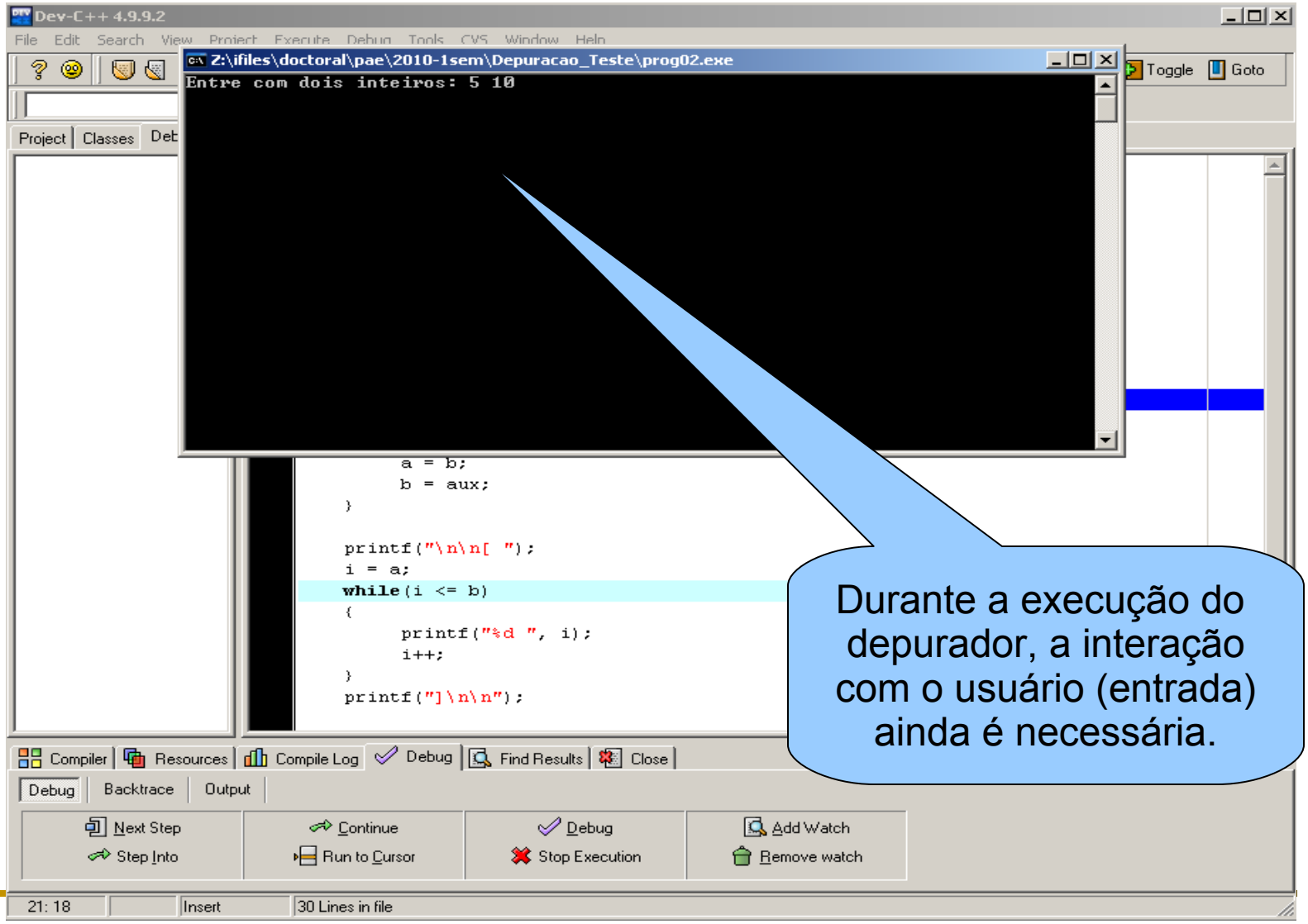
# Depuração: Dev-C++(passo 2)



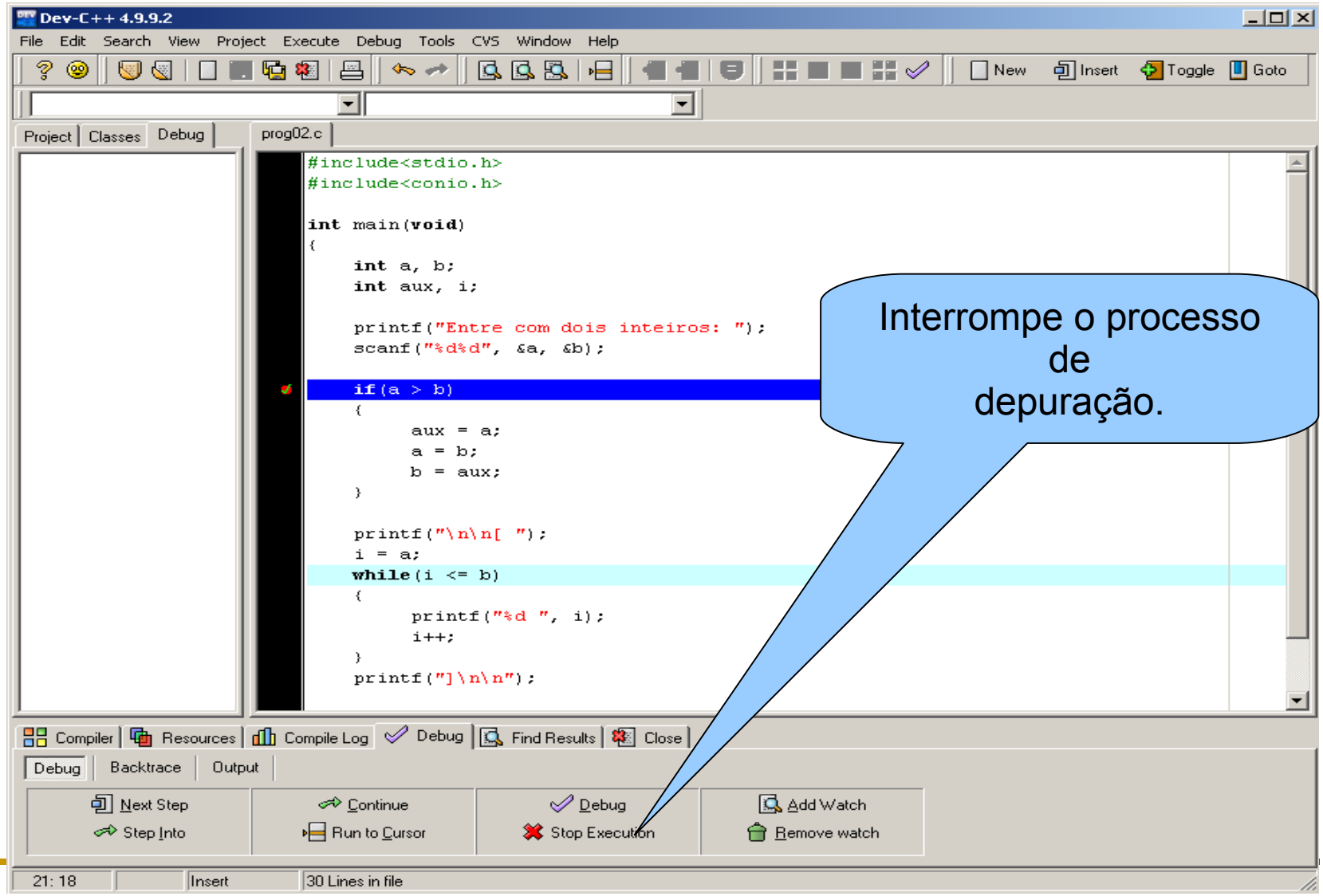
# Depuração: Dev-C++(passo 3)



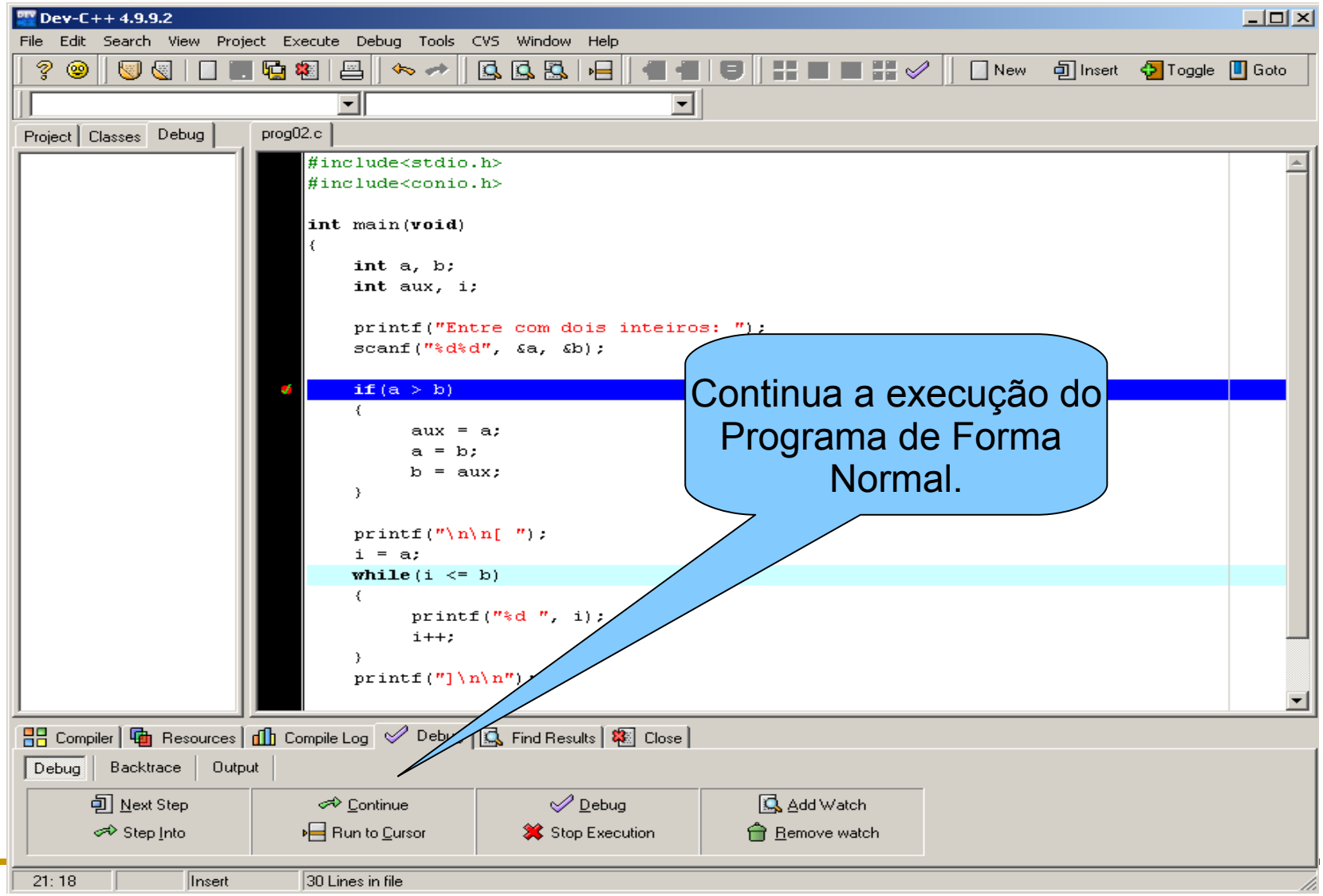
# Depuração: Dev-C++(passo 4)



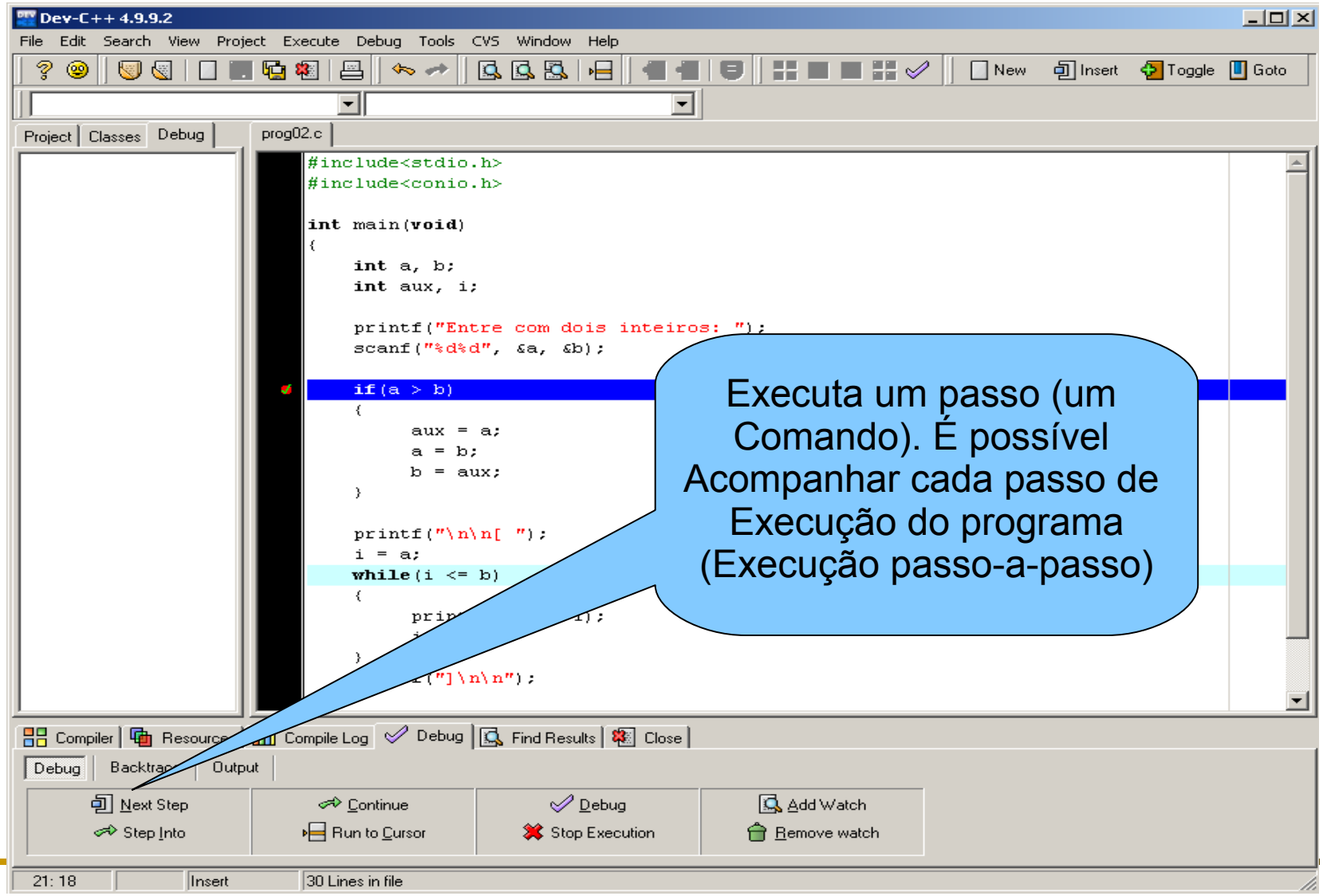
# Depuração: Dev-C++(opções)



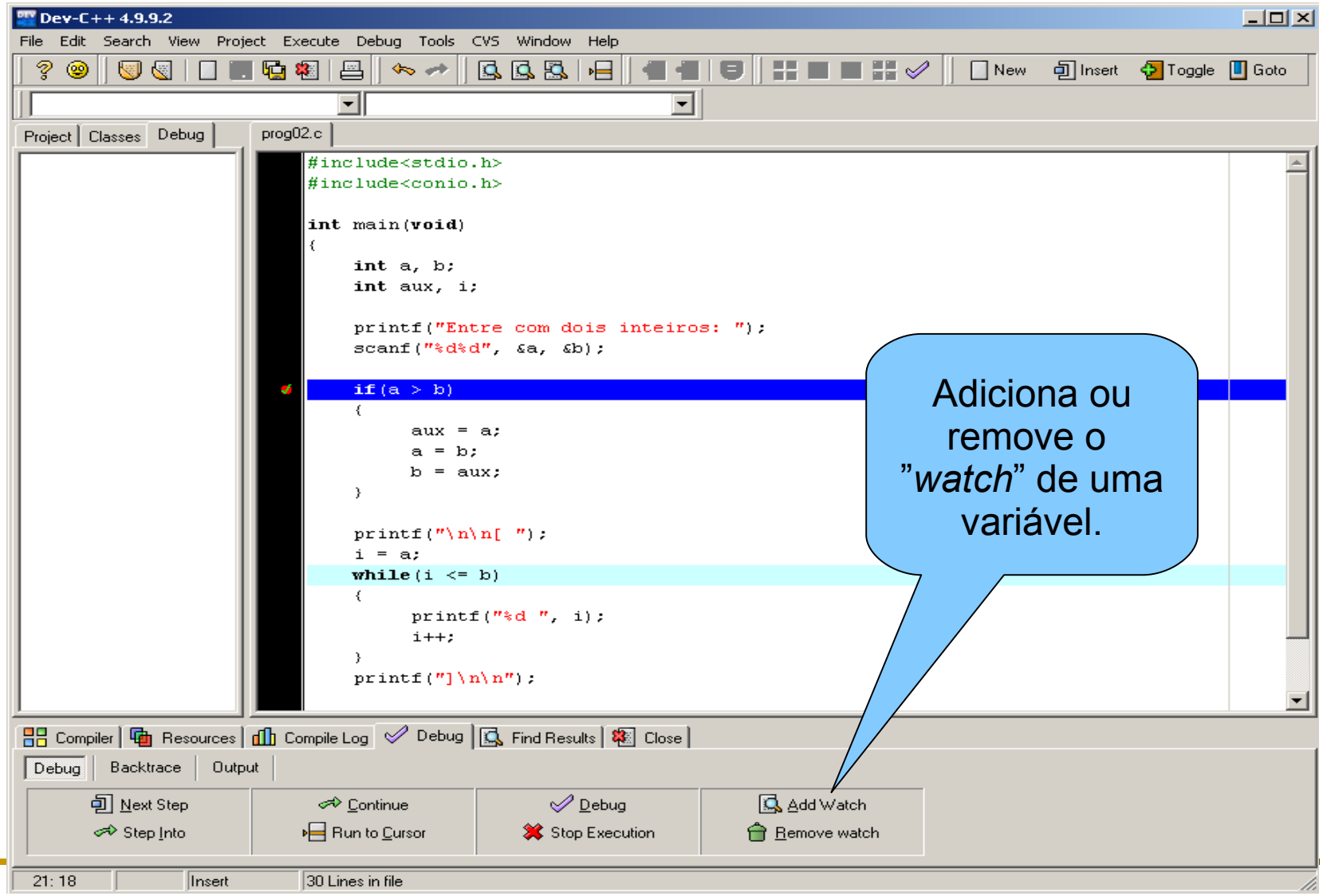
# Depuração: Dev-C++(opções)



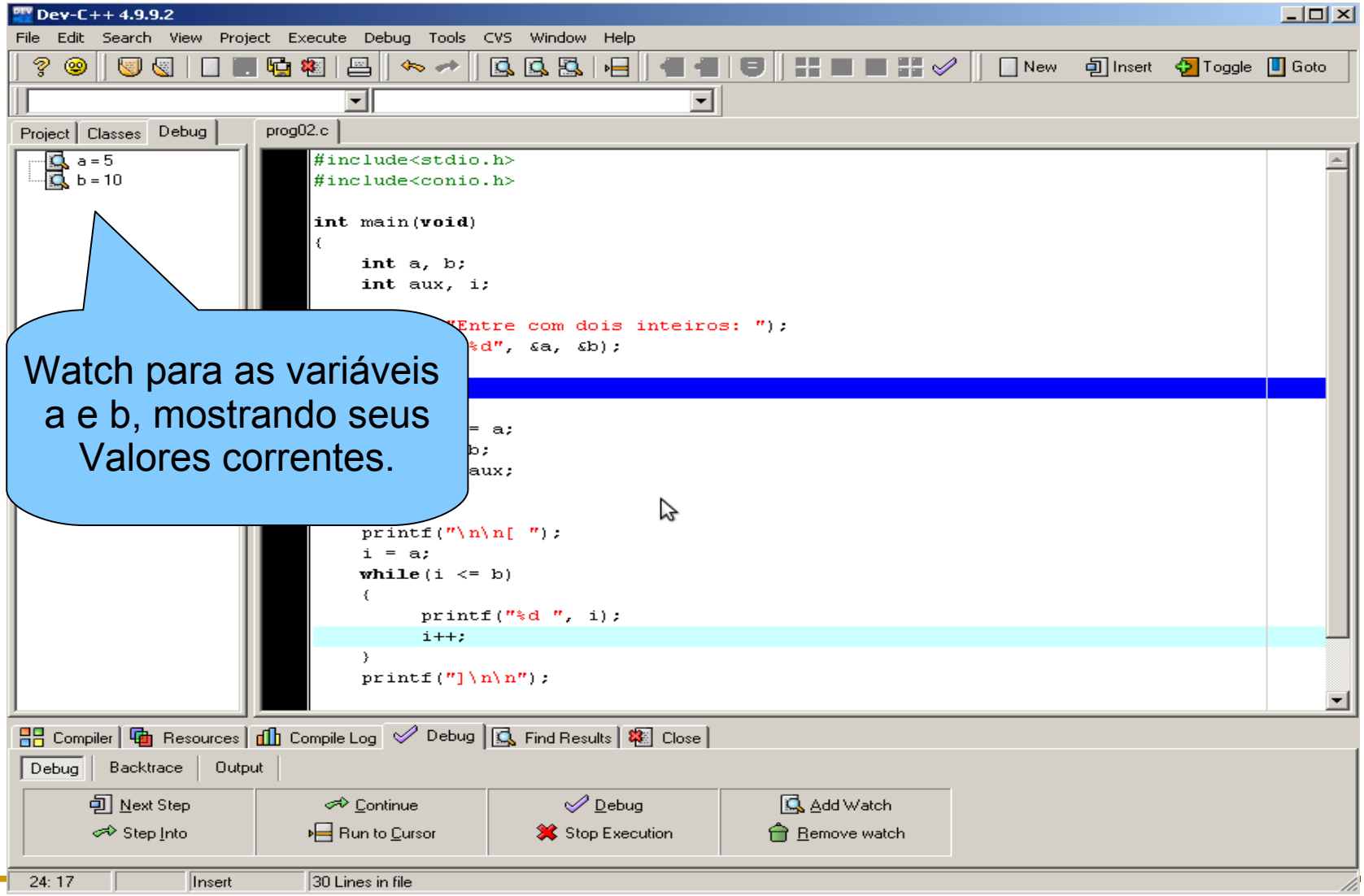
# Depuração: Dev-C++(opções)



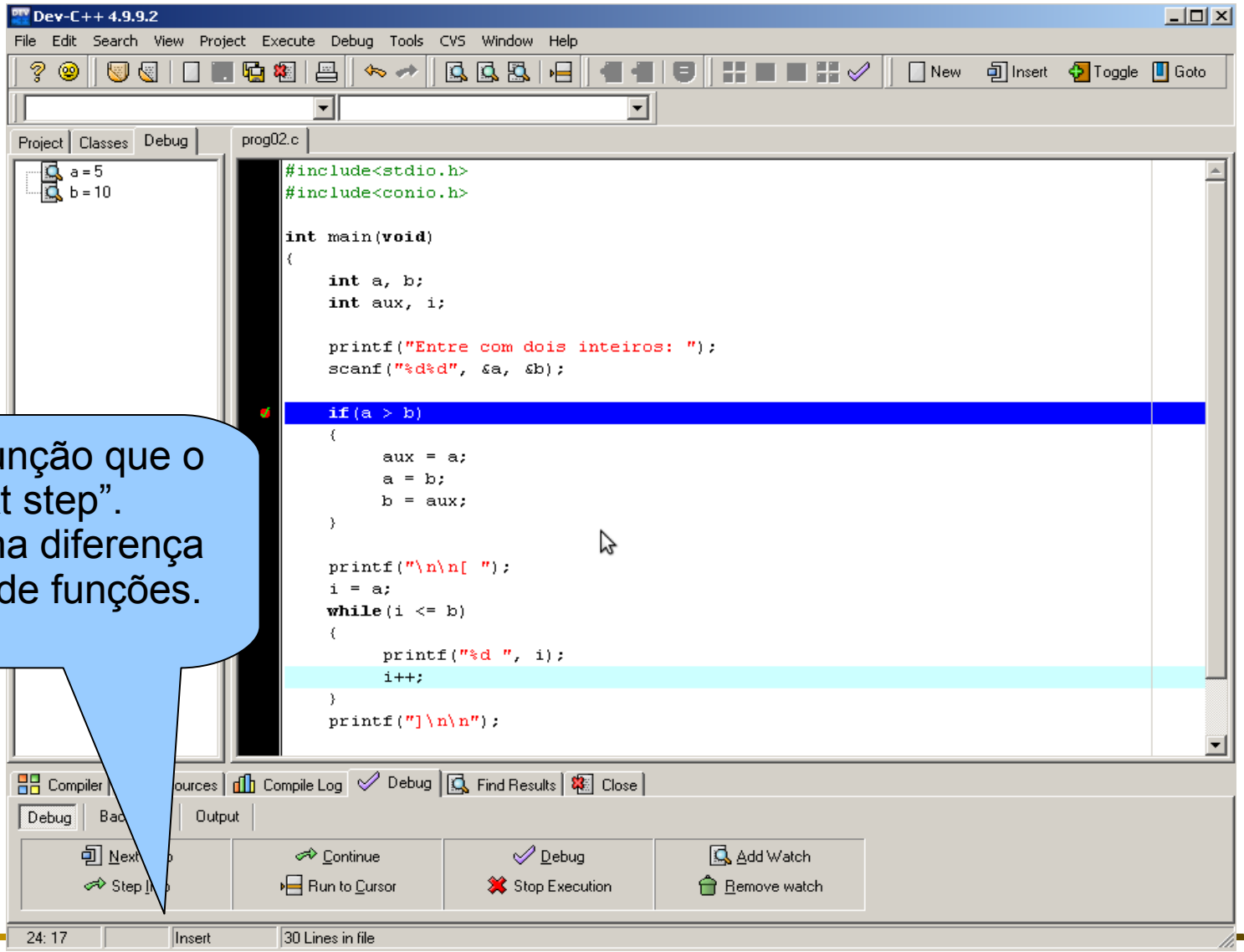
# Depuração: Dev-C++(opções)



# Depuração: Dev-C++(opções)

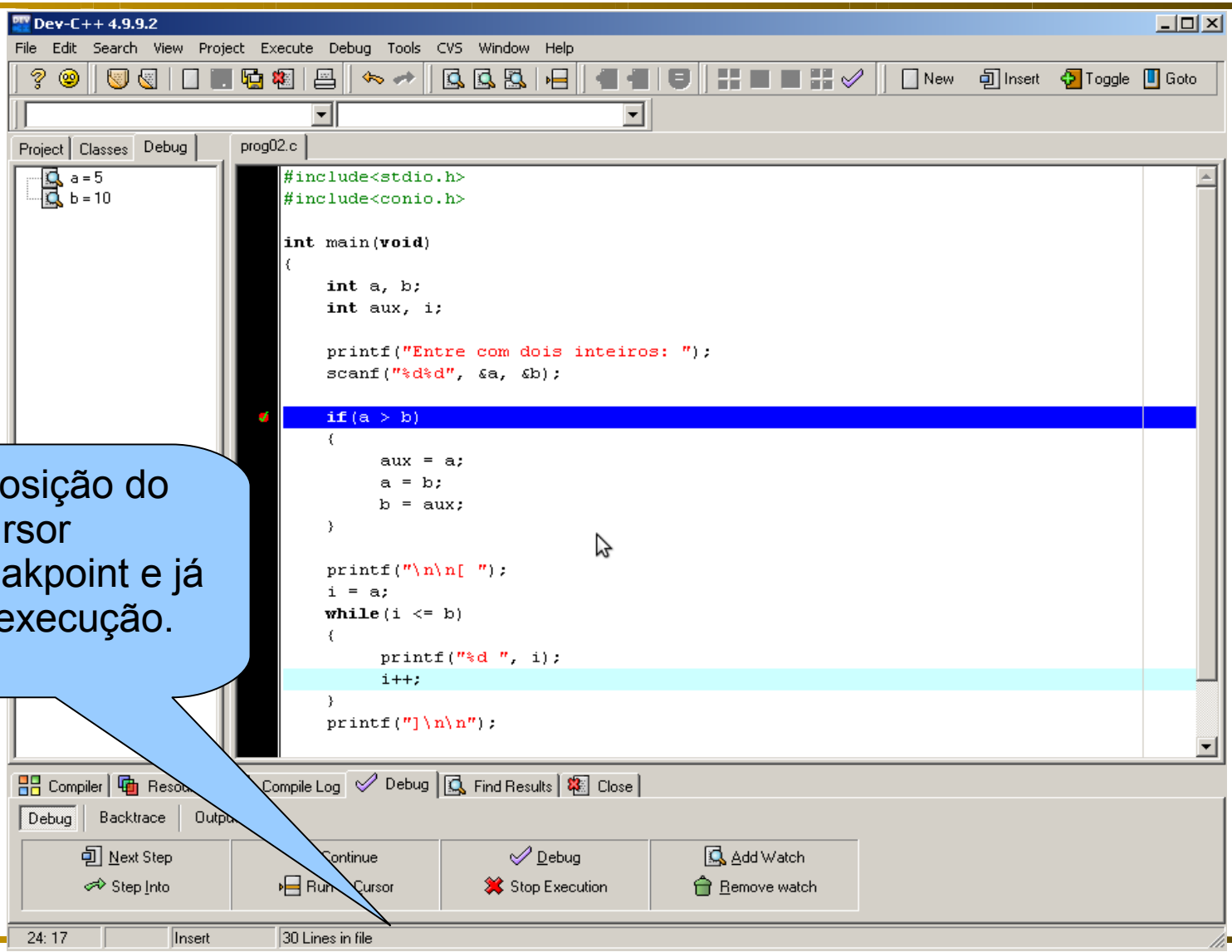


# Depuração: Dev-C++ (Opções)




Mesma função que o ”next step”.
Existe uma diferença no caso de funções.

# Depuração: Dev-C++ (Opções)




Usa a posição do cursor
Como breakpoint e já Inicia a execução.

# Depuração: Dev-C++

- Exemplo: faça um programa que receba como entrada dois inteiros a e b e imprima o intervalo:
  - [a, b] se a <= b
  - [b, a], caso contrário (b < a).
- Pegue o código-fonte no STOA (prog02.c)

# Depuração: Dev-C++

- Exercício 01: utilizando o programa 02, faça:
  - A) Coloque um *breakpoint* na linha ”int a, b;” e faça uma execução passo-a-passo para o caso de teste
    - <5, 10; [ 5 6 7 8 9 10 ] >
  - B) Coloque um *breakpoint* na linha ”int a, b;” e faça uma execução passo-a-passo para o caso de teste
    - <11, 7; [ 7 8 9 10 11 ] >
  - Neste exercício, adicione o *watch* para as variáveis a, b, i, aux.

# Depuração: Dev-C++

- Exercício 02: encontre os 3 defeitos no programa 3 (prog03.c) e corrija-os.
  - Leia atentamente a descrição do que o programa deve fazer.
  - Descreva, na forma de comentários no código, os casos de teste usados para revelar os defeitos.
  - Como o depurador ajudou/ajudaria a identificar esses defeitos?

# Depuração: gcc + gdb (Linux)

- GCC (Dev-C++)
- gdb – linha de comando
- Utilize os terminais de comando do linux
  - Konsole, Gnome terminal
- Compilando o programa com informação de depuração (-g)

```
>> gcc -g myprogram.c -o myprogram <enter>

>> ./myprogram <enter>

>> gdb myprogram <enter>
```

# Depuração: gcc + gdb (Linux)

- gdb – linha de comando

```
(gdb) digite comandos aqui <enter>
```

| Comando | Descrição |
|---|---|
| q | Sai do gdb. |
| r | Executa o programa do início. |
| l | Lista porções do código. |
| b *numero* | Coloca um breakpoint na linha *numero* |
| n | Executa a próxima instrução. |
| p *var* | Imprime o valor atual da variável *var.* |
| c | Continua a execução do programa sem paradas. |

Universidade de São Paulo – São Carlos
Instituto de Ciências Matemáticas e de Computação

# Arquivos em C – Parte 1

Profa Rosana Braga

(adaptado de material da profa
Silvana Maria Affonso de Lara)

1º semestre de 2010

# Arquivos em Disco

- Abrindo e Fechando um Arquivo
  - fopen, exit, fclose
- Lendo e Escrevendo Caracteres em Arquivos
  - putc, getc, feof
- Outros Comandos de Acesso a Arquivos
  - *Arquivos pré definidos*
  - ferror e perror, fgets, fputs, fread
  - fwrite, fseek, rewind, remove
- Fluxos Padrão
  - fprintf, fscanf

# Arquivos em Disco

- Abrindo e Fechando um Arquivo
  - fopen, exit, fclose

Aula de hoje

- Lendo e Escrevendo Caracteres em Arquivos
  - putc, getc, feof

- Outros Comandos de Acesso a Arquivos
  - *Arquivos pré definidos*
  - ferror e perror, fgets, fputs, fread
  - fwrite, fseek, rewind, remove

- Fluxos Padrão
  - fprintf, fscanf

# Abrindo e Fechando Arquivo

- O sistema de entrada e saída do ANSI C é composto por uma série de funções, cujos protótipos estão reunidos em <stdio.h>

- Todas estas funções trabalham com o conceito de "ponteiro de arquivo". Podemos declarar um ponteiro de arquivo da seguinte maneira:

```
FILE *p;
```

- p será então um ponteiro para um arquivo. É usando este tipo de ponteiro que vamos poder manipular arquivos no C.

# Abrindo e Fechando Arquivo

**fopen**

função de abertura de arquivos. Seu protótipo é:

```
FILE *fopen (char *nome_do_arquivo, char *modo);
```

- O nome_do_arquivo determina qual arquivo deverá ser aberto. Este nome deve ser válido no sistema operacional que estiver sendo utilizado. O modo de abertura diz à função fopen() que tipo de uso vai se fazer do arquivo.

# Abrindo e Fechando Arquivo

| Modo | Significado |
|---|---|
| “r” | Abre um arquivo texto para leitura. O arquivo deve existir antes de ser aberto. |
| “w” | Abrir um arquivo texto para gravação. Se o arquivo não existir, ele será criado. Se já existir, o conteúdo anterior será destruído. |
| “a” | Abrir um arquivo texto para gravação. Os dados serão adicionados no fim do arquivo ("*append*"), se ele já existir, ou um novo arquivo será criado, no caso de arquivo não existente anteriormente. |
| “rb” | Abre um arquivo binário para leitura. Igual ao modo "r" anterior, só que o arquivo é binário. |

# Abrindo e Fechando Arquivo

| Modo | Significado |
|---|---|
| “wb” | Cria um arquivo binário para escrita, como no modo "w" anterior, só que o arquivo é binário. |
| “ab” | Acrescenta dados binários no fim do arquivo, como no modo "a" anterior, só que o arquivo é binário. |
| “r+” | Abre um arquivo texto para leitura e gravação. O arquivo deve existir e pode ser modificado. |
| “w+” | Cria um arquivo texto para leitura e gravação. Se o arquivo existir, o conteúdo anterior será destruído. Se não existir, será criado. |

# Abrindo e Fechando Arquivo

| Modo | Significado |
|---|---|
| “a+” | Abre um arquivo texto para gravação e leitura. Os dados serão adicionados no fim do arquivo se ele já existir, ou um novo arquivo será criado, no caso de arquivo não existente anteriormente. |
| “r+b” | Abre um arquivo binário para leitura e escrita. O mesmo que "r+" acima, só que o arquivo é binário. |
| “w+b” | Cria um arquivo binário para leitura e escrita. O mesmo que "w+" acima, só que o arquivo é binário. |
| “a+b” | Acrescenta dados ou cria uma arquivo binário para leitura e escrita. O mesmo que "a+" acima, só que o arquivo é binário |

# Exemplo

```
FILE *fp; /* Declaração de um arquivo */
fp = fopen ("exemplo.txt", "w");
    /* o arquivo se chama exemplo.txt e está
        localizado no diretório corrente */
if (!fp)
printf ("Erro na abertura do arquivo.");
```

- A condição !fp testa se o arquivo foi aberto com sucesso porque no caso de um erro a função *fopen()* retorna um ponteiro nulo (NULL).

# Função exit

**exit -** O prototipo da função exit() é:

void exit (int codigo_de_retorno);

- Para utilizá-la deve-se colocar um *include* para o arquivo de cabeçalho *stdlib.h*
- Esta função aborta a execução do programa. Pode ser chamada de qualquer ponto no programa e faz com que o programa termine e retorne, para o sistema operacional, o código_de_retorno.
- A convenção mais usada é que um programa retorne zero no caso de um término normal e retorne um número não nulo no caso de ter ocorrido um problema.
- A função exit() é importante em casos de abertura de arquivos, pois se o programa não conseguir a memória necessária para abrir o arquivo, a melhor saída pode ser terminar a execução do programa.

# Exemplo

```
#include <stdio.h>
#include <stdlib.h> /* Para a função exit() */
main (void) {
FILE *fp;
...
fp = fopen ("exemplo.txt",“w");
if (!fp) {
   printf ("Erro na abertura do arquivo. Fim de programa.");
   exit (1);
}
...
}
```

# Abrindo e Fechando Arquivo

**fclose**

- Quando acabamos de usar um arquivo que abrimos, devemos fechá-lo. Para tanto usa-se a função *fclose()*:

```
int fclose (FILE *fp);
```

- O ponteiro fp passado à função *fclose()* determina o arquivo a ser fechado. A função retorna zero no caso de sucesso.

# Abrindo e Fechando Arquivo

- Fechar um arquivo faz com que qualquer caracter que tenha permanecido no "buffer" associado ao fluxo de saída seja gravado

*Mas, o que é este "buffer"?*

- Quando você envia caracteres para serem gravados em um arquivo, estes  caracteres são armazenados temporariamente em uma área de memória (o "buffer") em vez de serem escritos em disco imediatamente. Quando o "buffer" estiver cheio, seu conteúdo é escrito no disco de uma vez.

# Abrindo e Fechando Arquivo

- A razão para utilização desse buffer está relacionada à eficiência nas leituras e gravações de arquivos
  - Se, para cada caracter que fossemos gravar, tivéssemos que posicionar a cabeça de gravação em um ponto específico do disco, apenas para gravar aquele caracter, as gravações seriam muito lentas

- Assim estas gravações só são efetuadas quando houver um volume razoável de informações a serem gravadas ou quando o arquivo for fechado

- A função exit() fecha todos os arquivos que um programa tiver aberto.

# Escrevendo Caracteres em Arquivos

**putc**

- Escreve um caractere no arquivo
- Protótipo: **int** putc (int ch, FILE *fp);

**fscanf**

- Usado para leitura formatada de texto

Protótipo: **int** fscanf (FILE *fp,char *str,...);

**fprintf**

- Usado para gravação formatada de arquivos texto

Protótipo: **int** fprintf(FILE *fp,char *str,...);

# Exemplo

```
#include <stdio.h>
#include <stdlib.h>
void main()
{
FILE *fp;
char string[100];
int i;
if((fp = fopen("arquivo.txt","w")) == NULL) {
/* Arquivo ASCII, para escrita */
        printf( "Erro na abertura do arquivo");
        exit(0);
}
printf("Entre com a string a ser gravada no
arquivo:");
gets(string);
for(i=0; string[i]; i++) putc(string[i], fp);
   /* Grava a string, caractere a caractere */
fclose(fp);
}
```

# Exemplo

```
/* fprintf example */
#include <stdio.h>

int main ()
{
  FILE * pFile;
  int n;
  char name [100];

  pFile = fopen ("myfile.txt","w");
  for (n=0 ; n<3 ; n++)
  {
    puts ("please, enter a name: ");
    gets (name);
    fprintf (pFile, "Name %d [%-
10.10s]\n",n,name);
  }
  fclose (pFile);
  return 0;
}
```

# Exemplo

```
/* fscanf example */
#include <stdio.h>

int main ()
{
  char str [80];
  float f;
  FILE * pFile;

  pFile = fopen ("myfile.txt","w+");
  fprintf (pFile, "%f %s", 3.1416, "PI");
  rewind (pFile);
  fscanf (pFile, "%f", &f);
  fscanf (pFile, "%s", str);
  fclose (pFile);
  printf ("I have read: %f and %s \n",f,str);
  return 0;
}
```

# Depuração e Arquivos

- Exercício 03: encontre os 2 defeitos nos programa 4 e 5 (prog04.c e prog05.c) e  corrija-os.
  - Descreva, na forma de comentários no código, os casos de teste usados para revelar os defeitos.
  - Como o depurador ajudou/ajudaria a identificar esses defeitos?