

# LINGUAGEM DE PROGRAMAÇÃO II

---

© 2022, Gizelle Kupac Vianna (DECOMP/UFRRJ)

# BIBLIOTECAS DE FUNÇÕES

---

## Aula 6

# Bibliotecas em C

- Bibliotecas são amplamente utilizadas na linguagem C.
- Além da biblioteca padrão e das bibliotecas disponibilizadas pelo sistema operacional, o programador pode desenvolver suas próprias bibliotecas, para usar em seus projetos ou disponibilizá-las a terceiros.
- As bibliotecas podem ser construídas para ligação estática ou dinâmica (DLL) com o código executável que as utiliza.

# Biblioteca Padrão do C

- Existe uma grande quantidade de bibliotecas disponíveis para a linguagem C, algumas delas mais genéricas e outras mais específicas (processamento de imagens, serviços de rede, bases de dados, etc.)
- **C Standard Library** é o nome da biblioteca padrão da linguagem C.
- Ela possui inúmeras funções prontas e disponíveis para utilização, agrupadas em 24 arquivos de cabeçalho.
- A biblioteca foi escrita para fornecer um conjunto básico de operações, mantendo a portabilidade do C ANSI entre diversas plataformas.

Arquivo	Função
<b>&lt;assert.h&gt;</b>	Implementa ajuda na detecção de erros em versões de depuração de programas.
<b>&lt;complex.h&gt;</b>	Trata da manipulação de números complexos. – Até aqui
<b>&lt;ctype.h&gt;</b>	Funções para conversão de maiúsculas, minúsculas e outros tratamentos de caracteres.
<b>&lt;errno.h&gt;</b>	Teste de códigos de erro reportados pelas funções de bibliotecas.
<b>&lt;fenv.h&gt;</b>	Define várias funções e macros para tratar de exceções em variáveis do tipo ponto flutuante.
<b>&lt;float.h&gt;</b>	Define limites e precisão de variáveis de ponto flutuante.
<b>&lt;inttypes.h&gt;</b>	Trata de conversão precisa entre tipos inteiros.
<b>&lt;iso646.h&gt;</b>	Adiciona a possibilidade de programação usando a codificação de caracteres de acordo com a ISO646.
<b>&lt;limits.h&gt;</b>	Constantes de propriedades específicas de implementação da biblioteca de tipos inteiros, como a faixa de números que pode ser representada (_MIN, _MAX).

Arquivo	Função
<b>&lt;locale.h&gt;</b>	Especifica constantes de acordo com a localização específica, como moeda, data, etc.
<b>&lt;math.h&gt;</b>	Funções matemáticas comuns em computação.
<b>&lt;setjmp.h&gt;</b>	Define as macros setjmp e longjmp, para saídas não locais e tratamento de exceções.
<b>&lt;signal.h&gt;</b>	Implementa definições para receber e fazer o tratamento de sinais.
<b>&lt;stdarg.h&gt;</b>	Acesso dos argumentos passados para funções com parâmetro variável.
<b>&lt;stdbool.h&gt;</b>	Trata da definição para tipo de dados booleano.
<b>&lt;stdint.h&gt;</b>	Padrões de definição de tipos de dados inteiros.
<b>&lt;stddef.h&gt;</b>	Padrões de definições de tipos.
<b>&lt;stdio.h&gt;</b>	Tratamento de entrada/saída.

Arquivo	Função
<b>&lt;stdlib.h&gt;</b>	Implementa funções para diversas operações, incluindo conversão, alocação de memória, controle de processo, funções de busca e ordenação.
<b>&lt;string.h&gt;</b>	Tratamento de strings.
<b>&lt;tgmath.h&gt;</b>	Implementa facilidades para utilização de funções matemáticas.
<b>&lt;time.h&gt;</b>	Trata de tipos de data e hora.
<b>&lt;wchar.h&gt;</b>	Tratamento de caracteres para suportar diversas línguas.
<b>&lt;wctype.h&gt;</b>	Contém funções para classificação de caracteres <i>wide</i> .

# CRIANDO BIBLIOTECAS

---



# Arquivos de Cabeçalho (header)

- A criação de bibliotecas em C está atrelada à criação dos arquivos de cabeçalho ou headers (.h).
- Um arquivo de cabeçalho é um arquivo muito similar aos programas que vimos até agora. A diferença é que um arquivo header não contém a função `main()`, pois ele não gera um código executável.
- Os arquivos de cabeçalho são identificados pela extensão .h
- Ao incluir um arquivo de cabeçalho que não seja da biblioteca padrão do C, trocamos os sinais de `<>` por aspas duplas, como em:
  - `#include "utils.h"`

# Duplicação de includes

- Um problema que pode ocorrer ao usarmos includes é carregar a mesma mais de uma vez, por exemplo, quando o programa principal e uma include que este utiliza adicionam a mesma biblioteca.
- Isso acontece porque, quando linkamos uma biblioteca em um arquivo fonte, através da diretiva `#include`, o conteúdo do arquivo de cabeçalho é copiado exatamente neste ponto.
- Então, imagine que sua include chama a include `<stdio.h>` e o seu programa principal também. Haverá duas cópias da `stdio.h` no código, com as funções declaradas mais de uma vez.
- Durante a compilação, ocorrerá um erro de declaração duplicada.

# Guarda de Cabeçalho

- Para evitar a duplicação de código, usamos a guarda de cabeçalho.
- Para criar a guarda de cabeçalho usamos um comando de seleção especial para verificar se a include já foi adicionada. Esse comando deve ser inserido dentro das includes, e não no programa principal.
- Sintaxe:

```
#ifndef identificador
#define identificador

...
(corpo da include)
...
#endif
```

# Guarda de Cabeçalho

- No código anterior, *identificador* é um símbolo que identifica o trecho de código que será incluído no programa. A diretiva `#ifndef` verifica se o símbolo *identificador* já está definido. Se não estiver definido, o conteúdo até o `#endif` será incluído pelo compilador.
- Algumas observações sobre a escolha do identificador ajudam a evitar problemas inesperados:
  - Como o nome precisa ser exclusivo no contexto global, o recomendado é usar nomes longos o suficiente (ex: PROG, ou CAB, ou qualquer outro nome curto pode entrar em conflito com os nomes de variáveis ou funções, por exemplo).
  - Costuma-se usar maiúsculas para esses identificadores. Não é uma regra, mas é um estilo comum de programação.
  - Outro padrão informal é cercar o nome por dois sublinhados consecutivos.
  - Finalmente, o mais comum é usar o mesmo nome do arquivo .h, em letras maiúsculas e cercado por sublinhados duplos.

# Guarda de Cabeçalho - Exemplo

- `#ifndef __UTILS__`
- `#define __UTILS__`
- `...` (funções)
- `...`
- `#endif`

# Exemplo: utils.h

```
#ifndef __UTILS__
#define __UTILS__

char * maiusculas (char * frase) {
    int i = 0;
    while (frase[i] != '\0') {
        if (frase[i] > 90)
            frase[i] = frase[i]-32;
        i++;
    }
    return frase;
}
```

```
char * concatena (char * frase1, char * frase2) {  
    char * frase;  
    int i = 0, j = 0;  
  
    while (frase1[i] != '\0') {  
        frase[i] = frase1[i];  
        i++;  
    }  
  
    while (frase2[j] != '\0') {  
        frase[i] = frase2[j];  
        i++;  
        j++;  
    }  
    frase[i] = '\0';  
    return frase;  
}
```

```
char * capitaliza (char * frase) {  
    int i = 0;  
    frase[0] = frase[0]-32;  
  
    while (frase[i] != '\0') {  
        frase[i] = frase[i];  
        if (frase[i] == ' ') {  
            if (frase[i+1] > 90)  
                frase[i+1] = frase[i+1]-32;  
            i++;  
        }  
        i++;  
    }  
    return frase;  
}
```

#endif



# Exemplo: arquivo.c

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

```
#include "utils.h"
```



Includes proprietárias são  
incluídas entre aspas!

```
main() {
```

```
    char frase[30];
```

```
    int i = 0;
```

```
    printf("\nEntre com uma frase de ate 30 caracteres: \n");  
    gets(frase);
```

```
    printf("\n%s", frase);
```

```
    printf("\n%s", capitaliza(frase));
```

```
    printf("\n%s", maiusculas(frase));
```

```
    printf("\n%s", concatena(frase, frase));
```

```
}
```

# DEPURAÇÃO DE CÓDIGO

---

## Aula 6

# Depuração de Código

- Durante o desenvolvimento de um programa de computador, mesmo os mais simples, a chance de aparecerem erros no processo de compilação é de quase 100%.
- Mesmo depois de resolvidos os erros de sintaxe, ainda existe a chance enorme de acontecerem erros conceituais e lógicos.
- Erros de sintaxe são normalmente mais fáceis de resolver, até porque o compilador aponta para a posição do código onde o erro foi detectado.
- Porém, os erros conceituais e os de lógica podem ser bem mais traiçoeiros, por que podem ficar muito bem disfarçados dentro do código.

# Depuração de Código

- Algumas atitudes durante o desenvolvimento de um software podem ajudar:
  - Planeje o programa que você vai escrever, pensando bem nas funções que você vai usar.
  - Minimize o tamanho das funções, evitando funções que fazem muitas coisas de uma vez.
  - Mantenha o código claro, imagine sempre que outras pessoas terão que fazer a manutenção dele no futuro.
  - Evite escrever muitas linhas de código antes de verificar sua corretude.

# Depuração de Código

- Existem duas formas de acompanhar o funcionamento do programa:
  - Impressão de variáveis para rastrear a execução do programa;
  - Usar um depurador (debugger).
- A impressão das variáveis pode ser útil em programas menores, ou para debugar uma função menor.
- As ferramentas de depuração podem ser complicadas no início, e gasta-se um certo tempo para entender seus recursos, mas sempre será mais eficaz usar um depurador do que tentar rastrear o código manualmente.

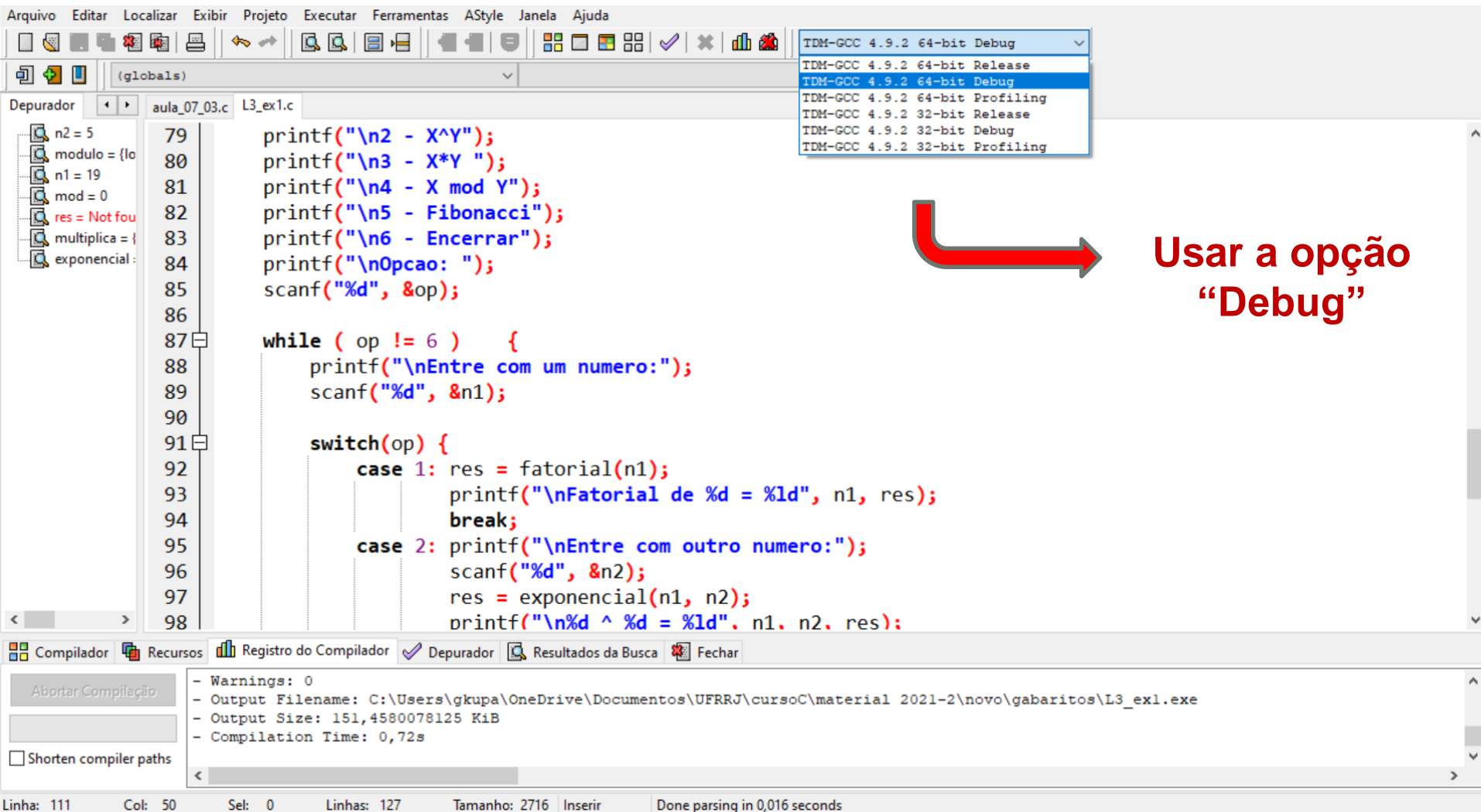
# Depuração de Código

- Mesmo que o programa rode sem erros, ainda assim é recomendado verificar, usando casos de testes, a integridade das respostas fornecidas.
- A tarefa de depuração, tanto quando feita a mão, quanto quando se usa uma ferramenta automática, requer a execução de um conjunto de entradas onde se sabe qual a saída esperada.
- São os chamados *casos de teste*.
- O ideal é aplicar os casos de teste no sistema de forma incremental, conforme se vai desenvolvendo o mesmo.

# Depuração de Código

- Durante a depuração usando um caso de teste é preciso definir:
  - O que você esperava que seu código fizesse?
  - O que ele fez?
  - Você consegue localizar em que comando (ou linha) de seu código o problema ocorreu?
- Um depurador permite que se percorra o código linha a linha, examinando toda e qualquer alteração em suas variáveis, ajudando a descobrir exatamente quando e como valores incorretos foram atribuídos a elas.
- Se a depuração for feita a mão, pode-se usar truques como imprimir mensagens a cada bloco suspeito, do tipo “chegou aqui”, “valor de  $i$  = ...”, etc.

# Usando o debugger do DevC



The screenshot shows the DevC++ IDE interface. The menu bar includes Arquivo, Editar, Localizar, Exibir, Projeto, Executar, Ferramentas, AStyle, Janela, and Ajuda. The toolbar contains icons for file operations, compilation, and debugging. The left sidebar shows the project structure with files like aula\_07\_03.c and L3\_ex1.c. The main editor displays the source code of L3\_ex1.c, which includes a while loop and a switch statement. The compiler options menu is open, showing various configurations. A red arrow points from the 'TDM-GCC 4.9.2 64-bit Debug' option to the text 'Usar a opção "Debug"'. The bottom status bar shows the current line and column.

Arquivo Editar Localizar Exibir Projeto Executar Ferramentas AStyle Janela Ajuda

(globals)

Depurador aula\_07\_03.c L3\_ex1.c

```
79 printf("\n2 - X^Y");
80 printf("\n3 - X*Y ");
81 printf("\n4 - X mod Y");
82 printf("\n5 - Fibonacci");
83 printf("\n6 - Encerrar");
84 printf("\nOpcao: ");
85 scanf("%d", &op);
86
87 while ( op != 6 ) {
88     printf("\nEntre com um numero:");
89     scanf("%d", &n1);
90
91     switch(op) {
92         case 1: res = fatorial(n1);
93                 printf("\nFatorial de %d = %ld", n1, res);
94                 break;
95         case 2: printf("\nEntre com outro numero:");
96                 scanf("%d", &n2);
97                 res = exponencial(n1, n2);
98                 printf("\n%d ^ %d = %ld", n1, n2, res);
```

TDM-GCC 4.9.2 64-bit Debug  
TDM-GCC 4.9.2 64-bit Release  
TDM-GCC 4.9.2 64-bit Debug  
TDM-GCC 4.9.2 64-bit Profiling  
TDM-GCC 4.9.2 32-bit Release  
TDM-GCC 4.9.2 32-bit Debug  
TDM-GCC 4.9.2 32-bit Profiling

Usar a opção "Debug"

Compilador Recursos Registro do Compilador Depurador Resultados da Busca Fechar

Abortar Compilação

Shorten compiler paths

Warnings: 0  
Output Filename: C:\Users\gkupa\OneDrive\Documentos\UFRRJ\cursoC\material 2021-2\novo\gabaritos\L3\_ex1.exe  
Output Size: 151,4580078125 KiB  
Compilation Time: 0,72s

Linha: 111 Col: 50 Sel: 0 Linhas: 127 Tamanho: 2716 Inserir Done parsing in 0,016 seconds



# Usando o debugger do DevC

Arquivo Editar Localizar Exibir Projeto Executar Ferramentas AStyle Janela Ajuda

Compilar F9  
Executar F10  
Compilar & Executar F11  
Recompilar Tudo F12

Depurador aula\_07\_03.c L3\_ex1.c

79 pri  
80 pri  
81 pri  
82 pri  
83 pri  
84 pri  
85 sca  
86  
87 whi  
88 numero:");  
89  
90  
91  
92 switch(op) {  
93 case 1: res = fatorial(n1);  
94 printf("\nFatorial de %d = %ld", n1, res);  
95 break;  
96 case 2: printf("\nEntre com outro numero:");  
97 scanf("%d", &n2);  
98 res = exponencial(n1, n2);  
99 printf("\n%d ^ %d = %ld", n1, n2, res);

Escolha a opção "Depurar", ou F5

Depurador

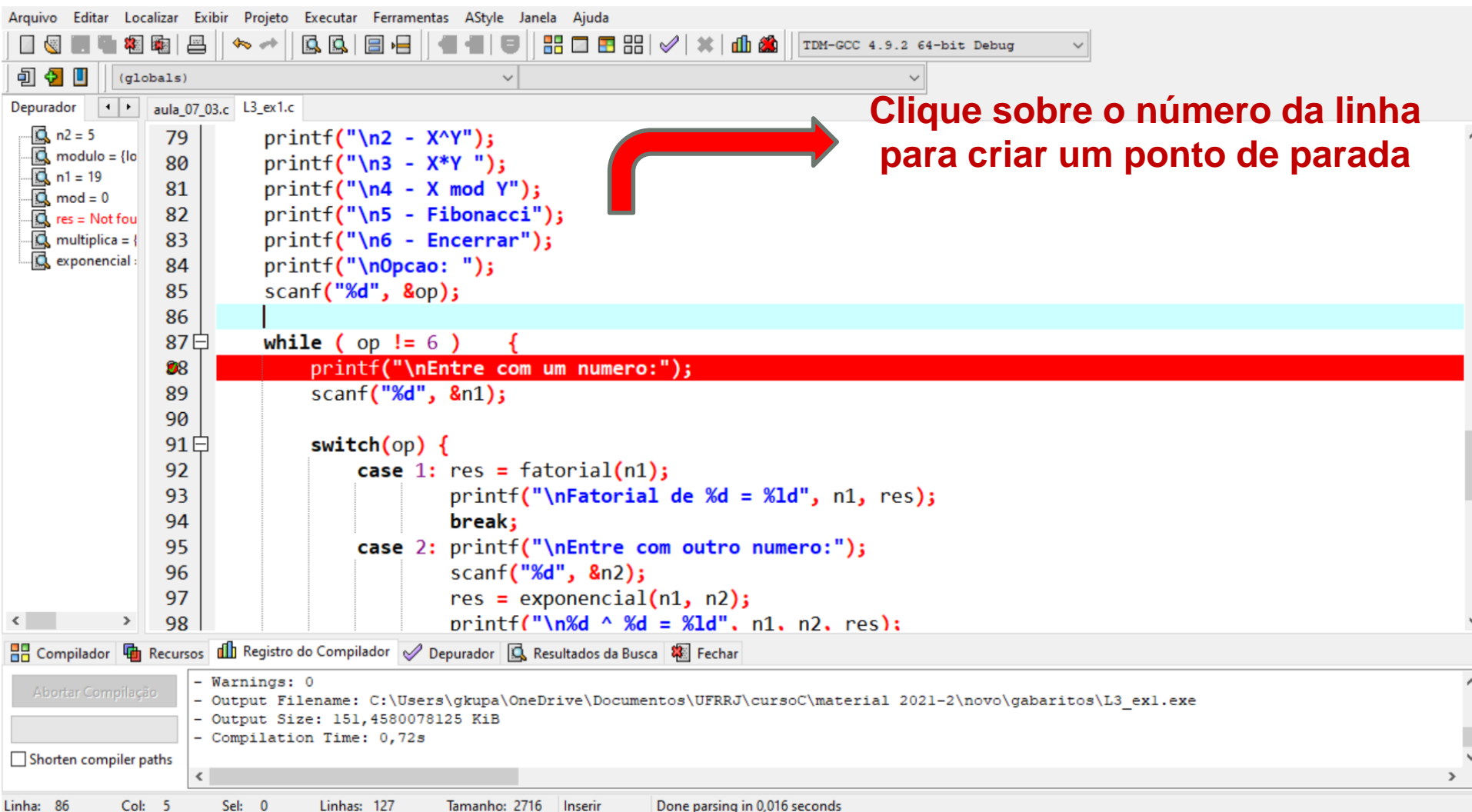
Abortar Compilação

Shorten compiler paths

Warnings: 0  
Output Filename: C:\Users\gkupa\OneDrive\Documentos\UFRRJ\cursoC\material 2021-2\novo\gabaritos\L3\_ex1.exe  
Output Size: 151,4580078125 KiB  
Compilation Time: 0,72s

Linha: 111 Col: 50 Sel: 0 Linhas: 127 Tamanho: 2716 Inserir Done parsing in 0,016 seconds

# Usando o debugger do DevC



Arquivo Editar Localizar Exibir Projeto Executar Ferramentas AStyle Janela Ajuda

(globals)

Depurador aula\_07\_03.c L3\_ex1.c

n2 = 5  
modulo = 10  
n1 = 19  
mod = 0  
res = Not found  
multiplica = 1  
exponencial = 1

```
79 printf("\n2 - X^Y");  
80 printf("\n3 - X*Y ");  
81 printf("\n4 - X mod Y");  
82 printf("\n5 - Fibonacci");  
83 printf("\n6 - Encerrar");  
84 printf("\nOpcao: ");  
85 scanf("%d", &op);  
86  
87 while ( op != 6 ) {  
88     printf("\nEntre com um numero:");  
89     scanf("%d", &n1);  
90  
91     switch(op) {  
92         case 1: res = fatorial(n1);  
93                 printf("\nFatorial de %d = %ld", n1, res);  
94                 break;  
95         case 2: printf("\nEntre com outro numero:");  
96                 scanf("%d", &n2);  
97                 res = exponencial(n1, n2);  
98                 printf("\n%d ^ %d = %ld", n1, n2, res);
```

Clique sobre o número da linha para criar um ponto de parada

Compilador Recursos Registro do Compilador Depurador Resultados da Busca Fechar

Abortar Compilação

Shorten compiler paths

Warnings: 0  
Output Filename: C:\Users\gkupa\OneDrive\Documentos\UFRRJ\cursoC\material 2021-2\novo\gabaritos\L3\_ex1.exe  
Output Size: 151,4580078125 KiB  
Compilation Time: 0,72s

Linha: 86 Col: 5 Sel: 0 Linhas: 127 Tamanho: 2716 Inserir Done parsing in 0,016 seconds