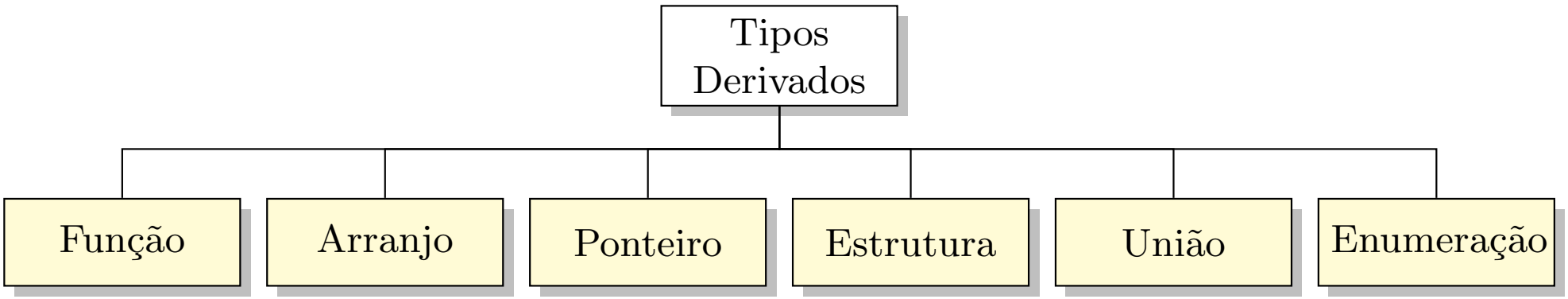
Funções

Alexsandro Santos Soares
prof.asoares@gmail.com

Universidade Federal de Uberlândia
Faculdade de Computação

Introdução

- Hoje estudaremos o primeiro tipo derivado em C, a tipo função.
- O tipo função é derivado de seu tipo de retorno.
- O tipo de retorno pode ser qualquer tipo válido em C, excetuando-se um arranjo e o tipo função.

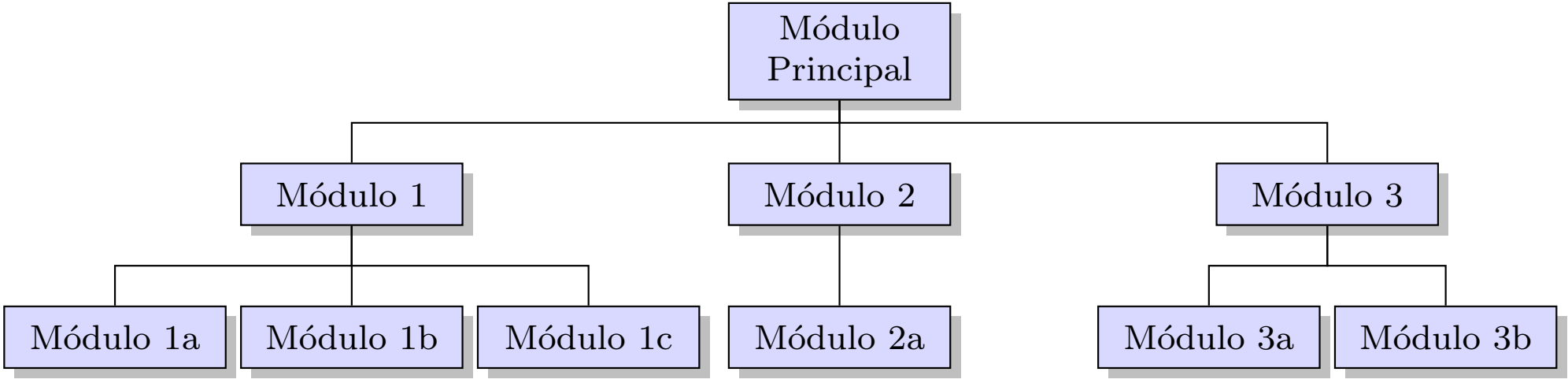


Programas estruturados

- Particionar um problema complexo em partes menores é uma prática comum em programação.
- Chamamos cada parte de um programa de módulo e o processo de subdividir um problema em partes de projeto top-down.
- No projeto top-down, um programa é divido em um módulo principal e nos módulos relacionados.
  - Cada módulo, por sua vez, é dividido em submódulos.
  - Essa divisão continua até que os submódulos consistam apenas de processos elementares, simples o suficiente para serem programados diretamente, sem quaisquer outras subdivisões.

Programas estruturados

- Em aulas anteriores já vimos o diagrama de estrutura modular (DEM) que é uma representação visual dos módulos.
- O DEM mostra o relacionamento entre cada cada módulo e seus submódulos.
- O DEM é lido de cima para baixo e da esquerda para a direita.
  - O módulo principal representa o conjunto completo de código que resolve o problema.
  - O módulo principal é denominado de módulo chamador pois ele possui submódulos.
  - Cada um dos submódulos é denominado de módulo chamado.



Programas estruturados

- A comunicação entre módulos em um DEM somente é permitida via módulo chamador.
  - Se o Módulo 1 precisar enviar dados para o Módulo 2, esses dados devem ser passados para o modulo chamador que, nesse caso, é o Módulo Principal.
- Nenhuma comunicação direta é permitida entre módulos que não possuam um relacionamento do tipo chamador-chamado.
  - Para o Módulo 1a enviar dados para o Módulo 3b, ele primeiro envia os dados para o Módulo 1 que, por sua vez, envia para o Módulo Principal, que os repassa para o Módulo 3 que finalmente envia para o Módulo 3b.
- A técnica usada para enviar dados para um módulo é conhecida como passagem de parâmetros.

Funções em C

- Em C, a ideia do projeto top-down é realizada por meio de funções.
- Um programa é composto de uma ou mais funções e apenas uma delas deve ser chamada de main (principal, em inglês).
- A execução do programa sempre começa e termina na *main*, mas ela pode chamar outras funções para realizar partes do serviço.
- Uma função em C, incluindo *main*, é um módulo independente que será chamado para realizar uma tarefa específica.
- A função chamada recebe o controle da função chamadora.
  - Quando a função chamada termina ela devolve o controle para a função chamadora.
  - A função chamada pode ou não retornar um valor para a chamadora.
  - A função *main* é chamada pelo sistema operacional e, ao terminar, devolve o controle para ele.

Funções em C

- De forma geral, o propósito de uma função é receber zero ou mais dados, operar sobre eles e retornar, no máximo, um dado.
- Ao mesmo tempo, uma função pode ter efeitos colaterais que é uma ação que resulta na mudança de estado de um programa.
- Como exemplo de efeitos colaterais temos:
  - Aceitar dados de fora do programa.
  - Enviar dados do programa para o monitor ou para um arquivo.
  - Alterar o valor de uma variável na função chamadora.

Vantagens do uso de funções

- O uso de funções em C, ou em qualquer outra linguagem, possui como principais vantagens:
  1. A fatoração os problemas em passos compreensíveis e gerenciáveis.
  2. Evita a repetição de código para a mesma tarefa no mesmo programa.
  3. A reutilização de código em outros programas.
     - Podemos criar bibliotecas de funções e usá-las em vários outros projetos.
     - Os compiladores de C, por exemplo, possuem bibliotecas para uma ampla gama de tarefas, tais como, `math.h` que contém muitas funções matemáticas e estatísticas.
  4. A proteção da dados via o mecanismo de dados locais:
     - Os dados locais consistem de dados descritos dentro da função.
     - Eles estão disponíveis apenas para a função e somente quando ela estiver em execução.
     - Quando a função não estiver sendo executada, os dados não estarão acessíveis.

Exemplo do uso de funções

```
11 #include <stdio.h>
12
13 /**
14  * @brief Multiplica dois números e retorna o produto.
15  *
16  * @param x primeiro número
17  * @param y segundo número
18  * @return O produto de x por y
19  */
20 int multiplica(int x, int y){
21     return x * y;
22 } // multiplica
23
24 int main(void){
25     int multiplicador = 0;
26     int multiplicando = 0;
27     int produto = 0;
28
29     printf("Digite dois inteiros: ");
30     scanf("%d%d", &multiplicador, &multiplicando);
31     produto = multiplica(multiplicador, multiplicando);
32     printf("O produto de %d por %d é%d\n",
33            multiplicador, multiplicando, produto);
34     return 0;
35 } // main
```

Exemplo do uso de funções

O código anterior ao ser compilado usando

```
gcc -std=c11 exemplo1.c -o exemplo1.exe
```

e executado produz como saída

```
> ./exemplo1.exe
Digite dois inteiros: 3 4
O produto de 3 por 4 é 12
```

Formas básicas de funções

- Podemos classificar a forma básica de uma função usando os argumentos e o valor de retorno.
  - A partir de agora chamaremos os argumentos de uma função de lista de parâmetros.
- As funções que não retornam valor algum são chamadas de funções void.
- Ao combinar os tipos de retorno e as listas de parâmetros temos as quatro categorias básicas de funções
  - funções *void* sem parâmetros;
  - funções *void* com parâmetros;
  - funções que retornam um valor mas não possuem parâmetros;
  - funções que retornam um valor e possum parâmetros.
- Discutiremos cada uma delas a seguir.

Funções void sem parâmetros

- Este tipo de função não recebe parâmetros e não retorna coisa alguma.
- Ela somente possui efeitos colaterais, tais como exibir uma mensagem, e é usada apenas por estes efeitos.

```
void cumprimento(void) {
   printf("Bom dia!");
   return;
} // cumprimento
```

- A chamada desta função requer o uso de parênteses:

```
cumprimento();
```

Funções void com parâmetros

- Este tipo de função recebe parâmetros mas não retorna coisa alguma.
- Ela somente possui efeitos colaterais, tais como exibir uma mensagem, e é usada apenas por estes efeitos.

```
void imprimeInteiro(int x) {
   printf("%d\n", x);
   return;
} // imprimeInteiro
```

- A chamada desta função também requer o uso de parênteses:

```
imprimeInteiro(10);
```

Exemplo do uso

O programa a seguir chama a função `imprimeInteiro` várias vezes

```
 9 #include <stdio.h>
10
11 /**
12  * @brief Imprime um inteiro na saída padrão.
13  *
14  * @param x número inteiro a ser impresso
15  */
16 void imprimeInteiro(int x){
17     printf("%d\n", x);
18     return;
19 } // imprimeInteiro
20
21 int main(void){
22     int a = 0;
23
24     a = 5;
25     imprimeInteiro(a);
26
27     a = 33;
28     imprimeInteiro(a);
29
30     return 0;
31 } // main
```

Funções sem parâmetros que retornam um valor

- Este tipo de função não recebe parâmetros mas retorna um valor.
- O uso mais comum desta forma de função é na leitura de dados do teclado ou de um arquivo, retornando a informação lida para a chamadora.

```
int leiaQuantidade(void){
   int x;

   printf("Digite a quantidade: ");
   scanf("%d", &x);
   return x;
} // leiaQuantidade
```

- Um exemplo de uso de `leiaQuantidade` na função `main`:

```
int main(void){
   int quant = 0;

   quant = leiaQuantidade();
   return 0;
} // main
```

Funções com parâmetros que retornam um valor

Abaixo temos um exemplo desta forma de função sendo definida e usada:

```
int quadrado(int x){
   return x * x;
} // quadrado

int main(void){
   int a;
   int b;

   scanf("%d", &a);
   b = quadrado(a);

   printf("%d ao quadrado é%d\n", a, b);
   printf("%d ao quadrado é%d\n", 5, quadrado(5));

   return 0;
}
```

Exemplo do uso com várias funções

Em programas grandes, a função **main** é escrita somente com chamadas para outras funções.

```
 9 #include <stdio.h>
10
11 /**
12  * @brief Lê um número inteiro da entrada padrão
13  *
14  * @return número lido
15  */
16 int leiaNum(void){
17    int x;
18
19    printf("Digite a quantidade: ");
20    scanf("%d", &x);
21    return x;
22 } // leiaNum
```

Exemplo do uso com várias funções

```
23
24
25 /**
26  * @brief Calcula o quadrado de um número
27  *
28  * @param x número
29  * @return o quadrado de x
30  */
31 int quadrado(int x){
32    return x * x;
33 } // quadrado
34
35
36 /**
37  * @brief Imprime um inteiro na saída padrão.
38  *
39  * @param x número inteiro a ser impresso
40  */
41 void imprimeInteiro(int x){
42     printf("%d\n", x);
43     return;
44 } // imprimeInteiro
```

Exemplo do uso com várias funções

```
45
46
47 int main(void){
48     int a = 0;
49     int b = 0;
50
51     a = leiaNum();
52     b = quadrado(a);
53     imprimeInteiro(b);
54
55     return 0;
56 } // main
```

Compilando e executando:

```
> gcc -std=c11 exemplo3.c -o exemplo3.exe
> ./exemplo3.exe
Digite a quantidade: 4
16
```

Exemplos de funções

- No que segue, apresentaremos mais quatro exemplos de programas onde há a definição e chamadas de funções:
  - O primeiro programa encontra e imprime o dígito menos significativo, aquele mais à direita, de um inteiro lido do teclado.
  - O segundo programa encontra e soma os dois dígitos menos significativos de um inteiro lido.
  - O terceiro lê um inteiro longo e o imprime com espaço após os primeiros três dígitos. Por exemplo, o inteiro 123456 serã impresso como 123456.
  - O quarto programa ilustra a construção de um menu de escolhas. Lá o sistema lerá um inteiro e dependendo da opção do usuário ele calculará várias funções envolvendo esse inteiro.

Programa 1 - Imprime dígito menos significativo

```c
#include <stdio.h>

/**
 * @brief Extrai o dígito menos significativo de um inteiro.
 *
 * @param num número inteiro
 * @return dígito menos significativo
 */
int primeiroDigito(int num){
    return (num % 10);
} // primeiroDigito

int main(void){
    int numero = 0;
    int digito = 0;

    printf("Digite um inteiro: ");
    scanf("%d", &numero);

    digito = primeiroDigito(numero);
    printf("\nO dígito menos significativo é: %d\n", digito);

    return 0;
} // main
```

Programa 1 - Imprime dígito menos significativo

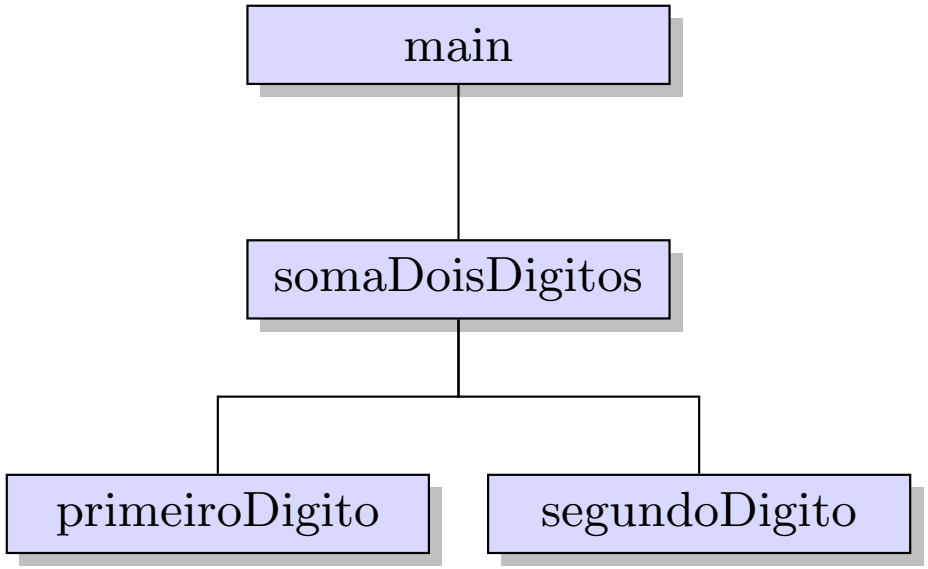
Compilando e executando:

```
> gcc -std=c11 exemplo4.c -o exemplo4.exe
> ./exemplo4.exe
Digite um inteiro: 27

O dígito menos significativo é:  7
```

Programa 2 - Extrai e soma os dois dígitos menos significativos

- O projeto deste programa é mostrado abaixo.



Programa 2 - Extrai e soma os dois dígitos menos significativos

```
 9 #include <stdio.h>
10
11 /**
12  * @brief Extrai o dígito menos significativo de um inteiro.
13  *
14  * @param num número inteiro
15  * @return dígito menos significativo
16  */
17 int primeiroDigito(int num){
18     return (num % 10);
19 } // primeiroDigito
```

Programa 2 - Extrai e soma os dois dígitos menos significativos

```
28 int segundoDigito(int num){
29     int resultado = 0;
30 
31     resultado = (num / 10) % 10;
32     return resultado;
33 } // segundoDigito
34 
35 /**
36  * @brief Soma os dois primeiros dígitos menos significativos de
37  *        um inteiro.
38  *
39  * @param num número inteiro
40  * @return a soma dos dois primeiros dígitos menos significativos num.
41  */
42 int somaDoisDigitos(int num){
43     int resultado = 0;
44 
45     resultado = primeiroDigito(num) + segundoDigito(num);
46     return resultado;
47 } // somaDoisDigitos
```

Programa 2 - Extrai e soma os dois dígitos menos significativos

```
50 int main(void){
51     int numero = 0;
52     int soma = 0;
53 
54     printf("Digite um inteiro: ");
55     scanf("%d", &numero);
56 
57     soma = somaDoisDigitos(numero);
58     printf("A soma dos dois dígitos menos significativos é: %d\n", soma);
59 
60     return 0;
61 } // main
```

Programa 2 - Extrai e soma os dois dígitos menos significativos

Compilando e executando:

```
> gcc -std=c11 exemplo5.c -o exemplo5.exe

> ./exemplo5.exe
Digite um inteiro: 23
A soma dos dois dígitos menos significativos é:  5

> ./exemplo5.exe
Digite um inteiro: 8
A soma dos dois dígitos menos significativos é:  8
```

Programa 3 - Formata inteiro longo

```c
/**
 * @file exemplo6.c
 * @brief Lê um número inteiro longo com no máximo 6 dígitos e o imprime
 *        com um espaço separando os três últimos dígitos.
 *
 * O número será impresso com zeros àesquerda caso seja menor que 100 000.
 *
 * @author Alexsandro Santos Soares
 * @date 30/04/2018
 * @bugs Nenhum conhecido.
 */
#include <stdio.h>
```

Programa 3 - Formata inteiro longo

```
14 /**
15  * @brief Separa um número inteiro em dois números, o último deles
16  *        contendo apenas três dígitos e imprime ambos separados por
17  *        espaço.
18  *
19  * @param num número inteiro
20  */
21 void imprimeComEspaco(long num){
22     int milhares = 0;
23     int centenas = 0;
24 
25     milhares = num / 1000;
26     centenas = num % 1000;
27 
28     printf("O número digitado é\t%03d %03d\n", milhares, centenas);
29     return;
30 } // imprimeComEspaco
```

Programa 3 - Formata inteiro longo

```
33 int main(void){
34     long int numero = 0;
35 
36     printf("Digite um inteiro com até 6 dígitos: ");
37     scanf("%ld", &numero);
38     imprimeComEspaco(numero);
39 
40     return 0;
41 } // main
```

Programa 3 - Formata inteiro longo

Compilando e executando:

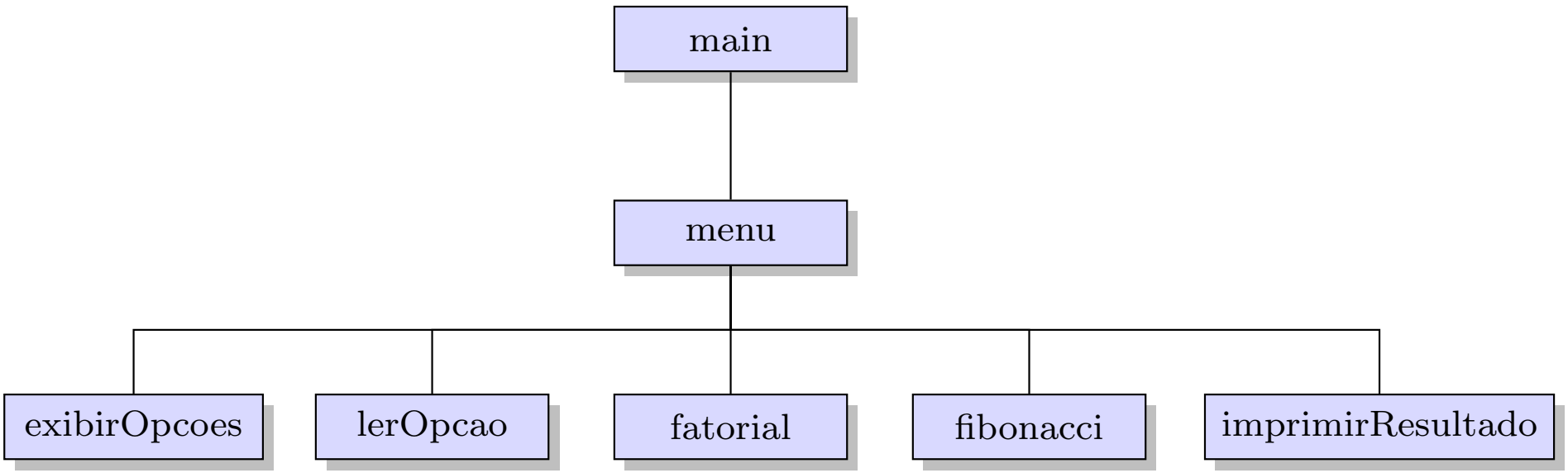
```
> gcc -std=c11 exemplo6.c -o exemplo6.exe

> ./exemplo6.exe
Digite um inteiro com até 6 dígitos: 123456
O número digitado é 123 456

> ./exemplo6.exe
Digite um inteiro com até 6 dígitos: 12
O número digitado é 000 012
```

Programa 4 - Criação de menu de opções

- O projeto deste programa é mostrado abaixo.



Programa 4 - Criação de menu de opções

```c
/**
 * @file exemplo7.c
 * @brief Cria um menu de opções, lê um inteiro e executa a função
 *        matemática escolhida sobre o inteiro lido.
 *
 * As funções matemáticas implementadas são:
 * 1 - fatorial
 * 2 - número de Fibonacci
 *
 * @author Alexsandro Santos Soares
 * @date 30/04/2018
 * @bugs Nenhum conhecido.
 */
#include <stdio.h>

#define OPCAO_MINIMA 1
#define OPCAO_MAXIMA 4
#define SAIR OPCAO_MAXIMA
```

```
26 unsigned long fatorial(unsigned long num){
27     unsigned long int i = 0; // contador de iterações
28     unsigned long int fat = 0; // acumula o valor do fatorial
29 
30     i = 0;
31     fat = 1;
32     while (i < num) {
33         i = i + 1;
34         fat = fat * i;
35     } // while
36 
37     return fat;
38 } // fatorial
```

```
47 unsigned long fibonacci(unsigned long n){
48     unsigned long int f1 = 0; // o penúltimo número de Fibonacci calculado.
49     unsigned long int f2 = 0; // o último número de Fibonacci calculado.
50     unsigned long int temp = 0; // temporário para guardar a soma.
51     unsigned long int i = 0;  // contador de iterações
52 
53     if (n == 0 || n == 1){
54         return n;
55     } else {
56         f1 = 0;
57         f2 = 1;
58         temp = 0;
59         i = 2;
60         while (i <= n) {
61             temp = f1 + f2;
62             f1 = f2;
63             f2 = temp;
64             i = i + 1;
65         } // while
66 
67         return f2;
68     } // else
69 } // fibonacci
```

```
75 void exibirOpcoes(void){
76     printf("\nOpções:\n\n");
77     printf("1 - fatorial\n");
78     printf("2 - n-ésimo número de Fibonacci\n");
79     printf("3 - ler um novo inteiro\n");
80     printf("4 - sair\n");
81     printf("\n");
82 } // exibirOpcoes
83 
84 /**
85  * @brief Lê uma opção válida informada pelo usuário
86  *
87  * @return um inteiro representando a opção escolhida
88  */
89 int lerOpcao(void){
90     int opcao = 0;
91 
92     printf("Digite uma opção: ");
93     scanf("%d", &opcao);
94     while ( !(OPCAO_MINIMA <= opcao && opcao <= OPCAO_MAXIMA) ){
95         printf("Opção inválida!\n\n");
96         printf("Digite uma opção: ");
97         scanf("%d", &opcao);
98     } // while
99 
100    return opcao;
101 } // lerOpcao
```

```
103 /**
104  * @brief Imprime o texto mostrando o resultado da operação
105  *        escolhida
106  *
107  * @param opcao número da opção escolhida
108  * @param num número sobre o qual a operação foi realizada
109  * @param resultado resultado da operação sobre num
110  */
111 void imprimirResultado(int opcao, long int num, long int resultado){
112     printf("\n");
113     printf("-------------------------------------------------\n");
114     if (opcao == 1){
115         printf("Fatorial de %lu = %lu\n", num, resultado);
116     } else if (opcao == 2){
117         printf("%lu-ésimo número de Fibonacci = %lu\n", num, resultado);
118     } else if (opcao == 3){
119         printf("Novo inteiro lido = %lu\n", num);
120     } else {
121         printf("Fim.\n");
122     }
123 
124     printf("-------------------------------------------------\n");
125 
126     return;
127 } // imprimirResultado
```

```
135 void menu(unsigned long num){
136     int opcao = 0;
137     unsigned long resultado = 0;
138 
139     while (opcao != SAIR){
140         exibirOpcoes();
141         opcao = lerOpcao();
142 
143         if (opcao == 1){
144             resultado = fatorial(num);
145         } else if (opcao == 2){
146             resultado = fibonacci(num);
147         } else if (opcao == 3){
148             printf("Digite um inteiro: ");
149             scanf("%lu", &num);
150         } else {
151         } // else
152 
153         imprimirResultado(opcao, num, resultado);
154     } // while
155 
156     return;
157 } // menu
```

```
160 int main(void){
161     unsigned long int numero = 0;
162 
163     printf("Digite um inteiro: ");
164     scanf("%lu", &numero);
165     menu(numero);
166 
167     return 0;
168 } // main
```

Programa 4 - Uso

Compilando e executando:

```
> gcc -std=c11 exemplo7.c -o exemplo7.exe

> ./exemplo7.exe
Digite um inteiro: 5

Opções:

1 - fatorial
2 - n-ésimo número de Fibonacci
3 - ler um novo inteiro
4 - sair

Digite uma opção: 1

------------------------------------------------
Fatorial de 5 = 120
------------------------------------------------
```

Programa 4 - Uso

```
Opções:

1 - fatorial
2 - n-ésimo número de Fibonacci
3 - ler um novo inteiro
4 - sair

Digite uma opção: 2

------------------------------------------------
5-ésimo número de Fibonacci = 5
------------------------------------------------
```

Programa 4 - Uso

```
Opções:

1 - fatorial
2 - n-ésimo número de Fibonacci
3 - ler um novo inteiro
4 - sair

Digite uma opção: 3
Digite um inteiro: 6

------------------------------------------------
Novo inteiro lido = 6
------------------------------------------------
```

Programa 4 - Uso

```
Opções:

1 - fatorial
2 - n-ésimo número de Fibonacci
3 - ler um novo inteiro
4 - sair

Digite uma opção: 1

------------------------------------------------
Fatorial de 6 = 720
------------------------------------------------
```

Programa 4 - Uso

```
Opções:

1 - fatorial
2 - n-ésimo número de Fibonacci
3 - ler um novo inteiro
4 - sair

Digite uma opção: 2

------------------------------------------------
6-ésimo número de Fibonacci = 8
------------------------------------------------
```

Programa 4 - Uso

```
Opções:

1 - fatorial
2 - n-ésimo número de Fibonacci
3 - ler um novo inteiro
4 - sair

Digite uma opção: 4

------------------------------------------------
Fim.
------------------------------------------------
```

Para saber mais

- Forouzan, B. A and Gilbert, R. F. *Computer Science: a structured programming approach using C*. 3rd edition. Cengage Learning, 2007.

Fontes

- Forouzan, B. A and Gilbert, R. F. *Computer Science: a structured programming approach using C*. 3rd edition. Cengage Learning, 2007.