

# Programação de Software Básico

## Aula 2: Bibliotecas – data, hora e gráficos

### Apresentação

---

A modularização no desenvolvimento de programas é uma abordagem que permitiu a simplificação na reutilização de código em linguagens como C. As bibliotecas são a forma mais conhecida de distribuir esses códigos já criados por outros programadores, sendo fundamentais no desenvolvimento de programas em C.

Nesta segunda aula, trataremos do uso e criação de bibliotecas na linguagem C, tema que irá nos acompanhar durante esta disciplina. O uso de biblioteca para manipulação de data e hora no sistema, bem como o uso de elementos gráficos, nos dará uma ideia inicial de como acessar recursos do sistema com código já desenvolvido por terceiros.

### Objetivos

---

- Discutir sobre o uso de bibliotecas em C;
- Aplicar bibliotecas em programas;
- Usar bibliotecas em programas com data, hora e gráficos.

### Funções e bibliotecas

---

No desenvolvimento de programas, existe a possibilidade de usar funções escritas por outros programadores. Uma função é um conjunto de instruções que recebe entradas, faz cálculos específicos e produz uma saída.

E por que precisamos de funções?

A ideia de usar funções é reunir algumas tarefas comuns ou repetidas e criar um código para que, em vez de escrever as mesmas instruções repetidamente para entradas diferentes, possamos chamar a função. Assim, as funções nos ajudam a reduzir a redundância de código.


## Comentário

Se a funcionalidade é executada em vários locais do software, em vez de escrever o mesmo código repetidamente, criamos uma função e a chamamos em todos os lugares. Isso também ajuda na manutenção, pois precisamos mudar só em um local se fizermos alterações futuras na funcionalidade.

## Definição e declaração de uma função

Uma **definição de função** fornece o corpo, a coleção de instruções que define o que a função faz. A forma geral de uma definição de função na linguagem de programação C é a seguinte:

```
Tipo_de_retorno nome_da_função (lista de parâmetros) {
corpo da função
}
```

 Clique nos botões para ver as informações.

### Tipo de retorno

Uma função pode retornar um valor. O **Tipo\_de\_retorno** é o tipo de dado do valor que a função retorna. Algumas funções executam as operações desejadas sem retornar um valor. Nesse caso, o **Tipo\_de\_retorno** é a palavra-chave void.

### Nome da função

Este é o nome real da função. O nome da função e a lista de parâmetros, juntos, constituem a assinatura da função.

### Parâmetros

Um parâmetro é como um espaço reservado. Quando uma função é chamada, você passa um valor para o parâmetro. Esse valor é referido como parâmetro ou argumento real. A lista de parâmetros refere-se ao tipo, a ordem e ao número dos parâmetros de uma função. Parâmetros são opcionais; isto é, uma função pode não conter parâmetros.

O código abaixo de uma função chamada max() usa dois parâmetros, num1 e num2, e retorna o valor máximo entre os dois:

```
/* função retornando o máximo entre dois números */
int max (int num1, int num2) {
```

```
/* declaração de variável local */  
int resultado;  
  
if (num1 > num2)  
    resultado = num1;  
else  
    resultado = num2;  
  
return resultado;  
}
```

Uma função deve ser declarada, por regra, antes de ser usada. Uma **declaração de função** informa ao compilador sobre o nome de uma função e como chamá-la. O corpo real da função pode ser definido separadamente. Uma declaração de função tem as seguintes partes:

```
Tipo_de_retorno nome_da_função (lista de parâmetros);
```

Para a função definida acima, `max()`, a declaração da função é a seguinte:

```
int max (int num1, int num2);
```

Os nomes dos parâmetros não são importantes na declaração da função, apenas o tipo é necessário; portanto, também é uma declaração válida `max()`:

```
int max (int, int);
```

A declaração de função é necessária quando você define uma função em um arquivo de origem e chama essa função em outro arquivo. Nesse caso, você deve declarar a função na parte superior do arquivo que está chamando a função.

## Chamando uma função

Ao criar uma função C, você define o que a função deve fazer. Para usar uma função, você precisará chamá-la para executar a tarefa definida.

Quando um programa chama uma função, o controle do programa é transferido para a função chamada. Uma função chamada executa uma tarefa definida e, quando sua instrução de retorno é executada ou quando sua chave de encerramento de função é atingida, ela retorna o controle do programa de volta ao programa principal.

**Para chamar uma função, basta passar os parâmetros necessários junto com o nome da função e, se a função retornar um valor, você poderá armazenar o valor retornado.**

Vejamos como, usando o exemplo anterior da função max():

```
#include <stdio.h>

/* declaração de função */
int max (int num1, int num2);

int main () {
    /* definição de variável local */
    int a = 100;
    int b = 200;
    int ret;

    /* chamando uma função para obter o valor máximo */
    ret = max (a, b);

    printf ("O valor máximo é:% d \n", ret);

    return 0;
}

/* função retornando o máximo entre dois números */
int max (int num1, int num2) {

    /* declaração de variável local */
    int resultado;

    if (num1> num2)
        resultado = num1;
    else
        resultado = num2;

    return resultado;
}
```

## Importância das funções

Funções tornam o código modular. Considere um arquivo grande com muitas linhas de códigos. Torna-se realmente simples ler e usar o código se estiver dividido em funções.

Funções também proporcionam abstração. Podemos usar, por exemplo, as **funções de uma biblioteca** sem nos preocupar com o trabalho interno executado por ela.

# O que é uma biblioteca?

As **funções de uma biblioteca** são funções embutidas que são agrupadas e colocadas em um arquivo comum, chamado biblioteca. Cada função de uma biblioteca executa uma operação específica. Podemos usar essas funções da biblioteca para obter a saída predefinida, em vez de escrever nosso próprio código para obter essas saídas.

Na primeira aula, já usamos funções da biblioteca `stdio`<sup>1</sup> como `printf()` e `scanf()`, conforme no seguinte exemplo:

```
#include <stdio.h>

int main() {
    int num;
    printf("Informe um numero:");
    scanf("%d", &num);
    printf("\n Voce informou o numero %d", num);
}
```

Todas as funções da biblioteca **stdio** são declaradas no arquivo de cabeçalho `stdio.h`. No programa do exemplo, esse arquivo de cabeçalho é incluído usando o comando `#include <stdio.h>` para fazer uso das funções declaradas nessa biblioteca. A extensão `.h` do arquivo da biblioteca vem da palavra inglesa header (cabeçalho).

Quando incluimos arquivos de cabeçalho em nosso programa C usando a diretiva `#include`, todos os códigos C dos arquivos de cabeçalho são incluídos em nosso programa. Em seguida, este programa C é compilado pelo compilador e executado.

## Biblioteca-padrão do C

O conjunto da biblioteca-padrão do C é composto, ao todo, por 24 bibliotecas (`stdio` é uma delas), representadas por arquivos `.h` de cabeçalho. A quantidade de arquivos da biblioteca-padrão não é grande, pois a ideia por trás dessa biblioteca é fornecer apenas um conjunto básico de operações, de tal forma que a portabilidade do C ANSI (padrão definido pela American National Standards Institute) entre diversas plataformas seja relativamente simples.

Saiba mais

Conheça [os arquivos e a descrição das funções básicas](#) dessa biblioteca.

**Atenção!** Aqui existe uma videoaula, acesso pelo conteúdo online

## Processamento da biblioteca

### Como uma biblioteca é processada?

Para entender o processamento das bibliotecas pelo compilador, primeiro vamos recordar como as aplicações em C são criadas.

---

**1** Os arquivos de origem C são pré-processados. Diretivas de pré-processador, como `#define`, `#include`, são resolvidas.

---

**2** Os arquivos de código pré-processados são compilados em linguagem de montagem (assembly).

---

Os arquivos de montagem são montados em arquivos de objeto. Arquivos de objeto são binários, contendo código de idioma de máquina. Os arquivos de objetos são salvos no disco para processamento adicional. Por conveniência, vários arquivos de objeto podem ser copiados opcionalmente em um único arquivo de biblioteca. Um arquivo de biblioteca não é nada mais que uma coleção de arquivos de objeto.

**3** Arquivos de biblioteca podem ser criados, editados em objetos únicos. Quando o fornecimento de fontes não é desejado, é prática comum fornecer software como bibliotecas em arquivos de objeto. Por exemplo, se você colocar o código de nossa Função `max()` abaixo em um arquivo e compilar no Dev C++, haverá um erro no compilador, mas o arquivo de objeto `.o` será gerado e poderá ser chamado em outro código-fonte.

```
/* Função max */
int max (int num1, int num2) {

    / * declaração de variável local * /
    int resultado;

    if (num1> num2)
        resultado = num1;
    else
        resultado = num2;

    return resultado;
}
```

---

Todos os arquivos e bibliotecas de objetos e o arquivo de controle do vinculador (linker) são fornecidos na vinculação. Não há diferença entre fornecer os mesmos arquivos de objeto separadamente ou como parte de uma biblioteca. O vinculador resolve dependências, descarta objetos não referenciados e coloca tudo na localização de memória correta, com base nas informações no arquivo de controle do vinculador. O resultado é um arquivo executável ou uma imagem binária.

**4** Dessa forma as funções da biblioteca, funções escritas em C, são compiladas e fornecidas juntamente com outros objetos. A biblioteca-padrão C é uma coleção de funções-padrão definidas na especificação da linguagem C. A biblioteca-padrão é normalmente fornecida junto com o conjunto de ferramentas do compilador.

---

**A funcionalidade de alto nível das funções da biblioteca é padronizada, mas a implementação não é. A mesma função de biblioteca-padrão é implementada de forma completamente diferente em sistemas operacionais diferentes ou em uma aplicação para microcontrolador. Por exemplo, a função `printf()` envia dados para a tela, em uma aplicação para Windows, e para a porta serial, em uma aplicação para Arduino.**

## Biblioteca de data e hora (time.h)

---

Parte da biblioteca-padrão, a biblioteca de funções `time.h` é usada em C para interagir com rotinas de tempo do sistema e para mostrar saídas formatadas de tempo. Embora a manipulação de data e hora do C seja considerada confusa, cheia de experimentos fracassados e armadilhas, ela é importante, ainda assim, para entendermos o acesso a funções básicas do sistema que ainda poderão nos desafiar no futuro.

### Histórico da biblioteca

Internamente, o tempo dessas funções é representado como os segundos desde a meia-noite de 1º de janeiro de 1970 no meridiano de Greenwich, sem correção de segundos bissextos. Esse é o tempo contado em segundos, como se tivesse sido incrementado a cada segundo em dias de duração constante de 86.400 segundos cada desde então.

Às vezes, isso é descrito como horário UTC ou segundos UTC, porque é baseado no mesmo meridiano zero e usa o mesmo segundo do Tempo Universal Coordenado (UTC).

Durante a evolução da biblioteca, a partir de 1968, os comprimentos típicos de palavras de máquina mudaram duas vezes: De 16 para 32 e, depois, para 64 bits, o que é típico hoje e parece improvável que mude no futuro próximo.

### Comentário

Um problema mais sério é que os contadores Unix `time_t` de 32 bits serão desativados em 19 de janeiro de 2038. Espera-se que isso cause problemas para os sistemas Unix incorporados; é difícil prever sua magnitude, e podemos apenas esperar que o evento seja do mesmo tipo do que causou preocupação na virada de 2000.

### Estrutura da biblioteca `time.h`

O tipo de dados `struct tm` define uma struct (estrutura) para ser utilizada em funções que trabalham com data e hora.

Segue uma descrição da struct `tm`:

```
struct tm {  
    int tm_sec;           //representa os segundos de 0 a 59  
    int tm_min;           //representa os minutos de 0 a 59  
    int tm_hour;          //representa as horas de 0 a 24
```

```
int tm_mday;           //dia do mês de 1 a 31
int tm_mon;            //representa os meses do ano de 0 a 11
int tm_year;           //representa o ano a partir de 1900
int tm_wday;           //dia da semana de 0 (domingo) até 6 (sábado)
int tm_yday;           //dia do ano de 1 a 365
int tm_isdst;          //indica horário de verão se for diferente de zero
};
```

As principais funções são mostradas na tabela a seguir:

Função	Descrição
<b>setdate()</b>	Modifica a data do sistema.
<b>getdate()</b>	Obtém a hora da CPU.
<b>clock()</b>	Obtém a hora corrente do sistema.
<b>time()</b>	Obtém a hora corrente do sistema como estrutura. Esta chamada de sistema fornece a quantidade de segundos desde zero hora de 1º de janeiro de 1970.
<b>difftime()</b>	Fornece a diferença entre duas horas dadas.
<b>strftime()</b>	Modifica o formato da hora atual.
<b>mktime()</b>	Converte data e hora do formato tm para o formato time_t (inteiro longo).
<b>localtime()</b>	Recebe um tempo em segundos de uma variável do tipo time_t, converte para o tempo local, armazena os dados na struct e retorna um ponteiro para uma struct do tipo tm com os dados locais.
<b>gmtime()</b>	Converte um valor time_t para uma estrutura tm como o relógio universal (UTC).
<b>ctime()</b>	Retorna uma string que contém informações de data e hora.
<b>asctime()</b>	Converte tm para uma string no formato Www Mmm dd hh:mm:ss yyyy, em que: Www é o dia da semana; Mmm, o mês, em letras; dd, o dia do mês; hh:mm:ss, a hora; e yyyy, o ano.

O exemplo a seguir usa a função time() para acessar o número de horas desde a zero hora de 1º de janeiro de 1970.

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

int main ()
{
    time_t seg;
    seg = time (NULL);

    printf ("O número de horas desde 1º de janeiro de 1970 é %d \n", seg / 3600);
    return 0;
}
```



Para obter uma versão legível da hora local atual, você pode usar a função `ctime()`. Essa sequência é seguida por um caractere de nova linha (`\n`) e converterá o objeto `time_t` apontado pelo relógio em uma string contendo hora e data locais correspondentes, como no exemplo:

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

int main(void)
{
    time_t minha_hora;
    minha_hora = time(NULL);
    printf(ctime(&minha_hora));

    return 0;
}
```

Um exemplo usando as funções `time()`, `asctime()` e `localtime()` é dado a seguir:

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

#define LEN 150
int main ()
{
    char buf[LEN];
    time_t curtime;
    struct tm *loc_time;

    //obtem hora corrente do sistema
    curtime = time (NULL);

    //converte para a hora local
    loc_time = localtime (&curtime);

    //mostra hora e data no formato-padrão
    printf("%s", asctime (loc_time));

    strftime (buf, LEN, "Hoje eh %A, %b %d.\n", loc_time);
    fputs (buf, stdout);
    strftime (buf, LEN, "A hora eh %I:%M:%S %p.\n", loc_time);
    fputs (buf, stdout);

    return 0;
}
```

## Manipulando a estrutura de tempo

Também é possível manipular a struct `tm` de tempo e criar sua própria data e hora usando a função **`mktime()`**. Vejamos um exemplo (observe que o exemplo não alterará a data do sistema, ele só imprime uma nova hora e data.):

```
#include <time.h>

#include <stdio.h>

int main(void)
{
    struct tm str_time;
    time_t hora_do_dia;

    str_time.tm_year = 2019-1900;
    str_time.tm_mon = 7;
    str_time.tm_mday = 24;
    str_time.tm_hour = 10;
    str_time.tm_min = 3;
    str_time.tm_sec = 5;
    str_time.tm_isdst = 0;

    hora_do_dia = mktime(&str_time);
    printf(ctime(&hora_do_dia));

    return 0;
}
```

A saída desse código é:

```
Sat Aug 24 10:03:05 2019
```

## Usando a função difftime()

A função difftime() é muito útil, pois pode ser usada para medir o tempo de processamento de uma determinada parte do código. Por exemplo, abaixo medimos o tempo de um loop for com 500.000.000 de iterações que não está fazendo nada.

```
#include <stdio.h>

#include <time.h>

int main(void)
{
    time_t start,end;
    volatile long unsigned contador;

    start = time(NULL);

    for(contador = 0; contador < 500000000; contador++)
        ; /* Não executa nada */

    end = time(NULL);
    printf("O loop for usa %f segundos.\n", difftime(end, start));
    return 0;
}
```

# Fusos horários

Também é possível trabalhar com fusos horários diferentes usando a função `gmtime()` para converter o horário do calendário em UTC. Se você conhece a hora UTC, pode adicionar um valor, por exemplo, Brasília (BRA) -4 horas. Veja o seguinte exemplo:

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

#define BRA (-4)

int main ()
{
    time_t Tempo;
    struct tm *ptr_ts;

    time ( &Tempo );
    ptr_ts = gmtime ( &Tempo );

    printf ( "Hora de Brasilia: %2d:%02d\n",
            ptr_ts->tm_hour+BRA, ptr_ts->tm_min);
    return 0;
}
```

**Atenção!** Aqui existe uma videoaula, acesso pelo conteúdo online

## Criando sua própria biblioteca

Um programa em C pode ser dividido em vários arquivos-fonte, arquivos com extensão `.c`, que podem representar módulos do programa completo. Pode não se justificar para programas pequenos o uso de vários módulos.

### Comentário

No entanto, a modularização é muito útil em programas maiores, sendo uma técnica fundamental por facilitar a divisão de tarefas maiores e mais complexas, tornando-as tarefas menores, mais fáceis de implementar e de testar. Mais ainda, um módulo com funções C pode ser utilizado para compor vários outros programas, poupando tempo de programação.

A ideia da modularização, como visto, está ligada à criação de bibliotecas. Para ilustrar essa ideia, criaremos um programa que calcula o fatorial de um número fornecido pelo usuário, deixando a função `fatorial()`, porém, em um arquivo-fonte separado.

Criando um novo projeto no ambiente do Dev C++, devemos acrescentar dois arquivos: O `main.c`, arquivo-fonte principal; e o `biblioteca.c`, onde colocamos o código para o cálculo do fatorial do número fornecido. Abaixo, estão os dois códigos:

```
/* main.c */
#include <stdio.h>

int fatorial (int n);
```

```
int main() {  
  
    int num;  
    printf ("Digite um numero:");  
    scanf ("%d", &num);  
    printf("\nO fatorial de %d eh igual a %d", num, fatorial(num));  
  
    return 0;  
}  
  
/* Biblioteca.c */  
  
int fatorial (int n){  
  
    int i, resultado;  
    resultado = 1;  
    for(i=1; i <= n; i++)  
        resultado *= i;  
  
    return resultado;  
}
```

O arquivo biblioteca.c pode ser usado para compor outros programas (módulos) que queiram utilizar a função fatorial(). Para isso, esse outro módulo precisa conhecer as funções oferecidas por biblioteca.c por meio dos cabeçalhos (declarações) dessas funções. No exemplo dado, isso é feito pela repetição da declaração da função fatorial() no início do arquivo main.c.

Entretanto, para bibliotecas que ofereçam várias funções ou para módulos que queiram usar funções de muitas outras bibliotecas, essa repetição manual pode ficar muito trabalhosa e passível de erros.

Para resolver esse problema, toda biblioteca de funções em C costuma ter associada a ela um arquivo que contém apenas os cabeçalhos (declarações) das funções oferecidas e os tipos de dados que exporte, como define, typedef e struct. Esse arquivo de cabeçalho segue o mesmo nome da biblioteca à qual está associado, só que usa a extensão .h.

Dessa forma, podemos criar um arquivo biblioteca.h com o seguinte conteúdo:

```
/* Função oferecida pela Biblioteca biblioteca.h */  
/* Função fatorial, que retorna o valor fatorial do número passado como parâmetro */  
#ifndef Biblioteca_H  
#define Biblioteca_H  
int fatorial(int);  
#endif
```

## Observação 1

O uso de comentários explicando as funções, o que é muito útil para quem irá usar essa biblioteca.

O compilador não percebe que é o mesmo arquivo e não sabe ignorá-lo, a menos que você use essa sequência de definição especial. Se você está se perguntando por que o compilador pode tentar lê-lo duas vezes, a resposta é: Porque o `#include` funciona.

O `#include` é basicamente um copiar e colar. O que se diz em `#include "biblioteca.h"` é: Copie o conteúdo de biblioteca.h e cole tudo aqui. Isso significa que, se biblioteca.h for incluída mais de uma vez, você terá conjuntos duplicados de código. Por isso, você precisa dos protetores de inclusão: Para garantir que o compilador ignore as duplicatas.

Seu projeto agora deverá ter os arquivos main.c, biblioteca.h e biblioteca.c. Nos arquivos main.c e biblioteca.c, deve ser acrescentada a linha `#include "biblioteca.h"`. A linha de declaração da função `fatorial()`, `int fatorial(int)`, deve ser retirada do arquivo main.c, pois a declaração já faz parte do arquivo biblioteca.h.

Como você deve ter notado, há dois tipos de inclusão, e os dois têm um comportamento diferente. A diferença é onde o pré-processador procura o arquivo.

Se você usar a sintaxe `#include "biblioteca.h"`, o pré-processador procurará o arquivo na pasta local.

A sintaxe do colchete angular (`<>`), por outro lado, procura o arquivo em uma pasta de biblioteca especialmente designada e é normalmente usada para bibliotecas externas.

O que isso significa é que, para arquivos locais em seu projeto, você deve usar o estilo de citação `#include "biblioteca.h"` e, para bibliotecas externas, você deve usar o estilo de colchetes angulares, como `#include <stdio.h>`.

## Observação 2

o uso das diretivas para definir a função que vem a seguir, no caso a função `fatorial()`. A diretiva `#ifndef Biblioteca_H` diz “se Biblioteca\_H não estiver definida, defina-a e, em seguida, observe este código C”, que, implicitamente, significa “se Biblioteca\_H estiver definida, ignore o código abaixo”.

# Bibliotecas de funções gráficas

---

O padrão ANSI da linguagem C (C ANSI) não define rotinas gráficas. A utilização de rotinas gráficas demandou criações independentes, como o Borland Graphics Interface (BGI) e a bem difundida OpenGL, capaz de construir gráficos em 3D. Application Programming Interfaces (API) com diretivas para a programação de jogos, com funções para gráficos e controle via teclado e mouse, também surgiram para o ambiente C, como a Allegro ([//edcomjogos.dc.ufscar.br/tutoriais/tutorial\\_allegro.pdf](http://edcomjogos.dc.ufscar.br/tutoriais/tutorial_allegro.pdf)) e outras.

Note que falamos em `API`<sup>2</sup> para nos referir a essas bibliotecas, e há de ser feita mesmo essa diferenciação.

## Comentário

Nesta aula, vamos introduzir uma biblioteca gráfica básica para, na próxima, entrar em mais detalhes sobre o uso de elementos gráficos em programas.

## Biblioteca BGI (graphics.h)

**O graphics.h é uma biblioteca criada pela Borland e que já não é mais utilizada, mas permite uma visão geral no uso de elementos gráficos em C. Ela pode ser testada de forma simples no ambiente Windows com o antigo compilador Turbo C++.**

O compilador Turbo C++ do link (<https://getintopc.com/software/development/turbo-c-plus-plus-free-download/>) permite usar a antiga versão no ambiente Windows. Embora a janela aberta seja pouco amigável para usuários acostumados aos compiladores C mais novos, como Dev C++ e Code Blocks, muitas chamadas ao sistema permitidas no Turbo C++ são de grande valia ao aprendizado de programação de software básico. Algumas dessas chamadas serão indicadas em planos de aula subsequentes.

O programa a seguir pode ser testado no Turbo C++ baixado do link indicado no parágrafo anterior. Observe que o caminho C:\TurboC3\BGI deve conter os arquivos que na instalação são colocados em C:\TurboC++\Disk\TurboC3\BGI. É aconselhável usar um editor de texto, como o Notepad++, para editar e salvar o código em uma extensão de arquivo .c, salvando na pasta C:\TurboC++\Disk\TurboC3\BIN e, depois, abrir o arquivo no Turbo C++ para compilar e rodar.

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

main()
{
    int gd = DETECT, gm, left=100, top=100, right=200, bottom=200, x= 300, y=150, radius=50;

    initgraph(&gd, &gm, "C:\\TurboC3\\BGI");

    rectangle(left, top, right, bottom);
    circle(x, y, radius);
```

```
bar(left + 300, top, right + 300, bottom);
line(left - 10, top + 150, left + 410, top + 150);
ellipse(x, y + 200, 0, 360, 100, 50);
outtextxy(left + 100, top + 325, "Meu programa grafico");

getch();
closegraph();
return 0;
}
```

## Biblioteca graphics.h no Dev C++

**Atenção!** Aqui existe uma videoaula, acesso pelo conteúdo online

A biblioteca graphics.h foi adaptada para funcionar no compilador do Dev C++, que não permite chamadas ao sistema baseadas em DOS 16 bits, como o Turbo C++ pode fazer. Para importar a biblioteca para o Dev C++ Versão 5.11 deve-se seguir os seguintes passos:

### Passo 1

Baixar o driver adaptado em [Google Drive](#) (clique em download na página que se abre).



### Passo 2

Descompactar o arquivo "Graphics in Dev C++.zip".



### Passo 3

Copiar os arquivos 6-ConsoleAppGraphics.template e ConsoleApp\_cpp\_graph.txt (figura 1) para a pasta template dentro da pasta onde o Dev C++ está instalado: Em geral, nos arquivos de programa do Windows (C:\Program Files (x86)\Dev-Cpp\Templates).



### Passo 4

Copiar os arquivos graphics.h e winbgim.h para a pasta include do compilador que será usado (32 bits): Em geral, nos arquivos de programa do Windows (C:\Program Files (x86)\Dev-Cpp\MinGW64\x86\_64-w64-mingw32\include).



### Passo 5

Copiar o arquivo libbgi.a para a pasta lib e para a pasta lib32 do compilador que será usado (32 bits): Em geral, nos arquivos de programa do Windows (C:\Program Files (x86)\Dev-Cpp\MinGW64\x86\_64-w64-mingw32\lib) e (C:\Program Files (x86)\Dev-Cpp\MinGW64\x86\_64-w64-mingw32\lib32).



### Passo 6

Criar um projeto no Dev C++ com as opções Console Graphics e C++ Project, como mostrado na figura 2 (apesar de ser uma biblioteca de C, a opção C project não funciona).





## Passo 7

Escolher o compilador de 32 bits, como mostrado na figura 3 (TDM-GCC 4.9.2 32-bit Release).



Agora no arquivo main.cpp do projeto é possível escrever programas que acessem a biblioteca graphics.h. O exemplo a seguir realiza os mesmos gráficos do anterior feito para Turbo C++.

```
#include<graphics.h>

int main (){

    initwindow (800, 800);

    int left=100,top=100,right=200,bottom=200,x= 300,y=150,radius=50;

    rectangle(left, top, right, bottom);
    circle(x, y, radius);
    bar(left + 300, top, right + 300, bottom);
    line(left - 10, top + 150, left + 410, top + 150);
    ellipse(x, y + 200, 0, 360, 100, 50);
    outtextxy(left + 100, top + 325, "Meu programa grafico");

    getch();

}
```

## Atividade

1. Descreva como devem ficar os arquivos main.c, biblioteca.h e biblioteca.c para calcular, além do fatorial, o valor do cubo do número dado pelo usuário.
2. Com base na descrição da struct tm, crie um programa que mostre quantos dias se passaram no ano atual.

3. Descubra o que faz o programa abaixo. O que a função clock() fornece?

```
#include <stdio.h>
#include <time.h>
void fun()
{
    for (int i=0; i<20000000; i++)
    {
    }
}

int main()
{
    clock_t t;
    t = clock();
    fun();
    t = clock() - t;
    double x = ((double)t)/CLOCKS_PER_SEC;

    printf("%f \n", x);
    return 0;
}
```

4. Crie um projeto no Dev C++ para usar as funções da biblioteca graphics.h. Execute o programa a seguir e verifique o que ele faz.

```
#include<graphics.h>

int main (){
    initwindow (400, 400);
    char a[5];
    settextjustify(CENTER_TEXT, CENTER_TEXT);
    settextstyle(DEFAULT_FONT, HORIZ_DIR, 3);
    setcolor(RED);

    for (int i = 30; i >=0; i--)
    {
        sprintf(a, "%d", i);
        outtextxy(getmaxx()/2, getmaxy()/2, a);
        delay(1000);

        if (i == 0)
            break;
        cleardevice();
    }

    getch();
}
```

Com base nas bibliotecas de gráfico e tempo estudadas, altere o programa para que apresente, com letras gráficas, hora, minuto e segundo atuais no formato HH:MM:SS PM, durante 1 minuto.

5. Crie um programa em C para imprimir hora e data local.

Notas

Stdio<sup>1</sup>

(std de standard ou padrão; e io de input/output ou entrada/saída)

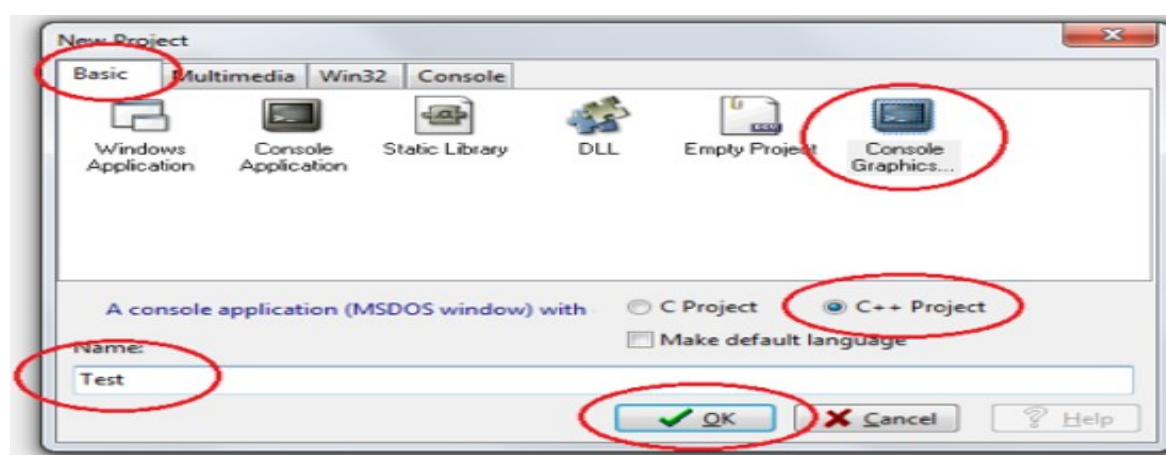
API<sup>2</sup>

A API é uma interface que orienta o programador quanto ao acesso a comandos, funções, protocolos ou objetos de um software ou plataforma. Ou seja, é a lista e a descrição das funções de uma biblioteca (ou de um aplicativo) que um programador pode chamar dentro do código. Basicamente, as APIs permitem que o desenvolvedor possa utilizar funcionalidades de aplicações já existentes no desenvolvimento de seu projeto.

(figura 1)

Nova pasta				
Nome	Data de modificaç...	Tipo	Tamanho	
Test Graphics.h	13/05/2019 17:20	Pasta de arquivos		
Test Winbgim.h	13/05/2019 17:20	Pasta de arquivos		
6-ConsoleAppGraphics.template	12/04/2019 12:17	Dev-C++ Templat...	1 KB	
ConsoleApp_cpp_graph.bt	12/04/2019 12:17	Documento de Te...	0 KB	
graphics.h	12/04/2019 12:17	C Header File	14 KB	
libbgia	12/04/2019 12:17	Arquivo A	55 KB	
winbgim.h	12/04/2019 12:17	C Header File	14 KB	

(figura 2)



## Referências

MANZANO, José Augusto N. G. **Linguagem C** – Acompanhada de uma Xícara de Café. 1. ed. São Paulo: Saraiva, 2015.

RAYMOND, Eric S. **Time, Clock, and Calendar Programming In C**. Disponível em: [http://www.catb.org/esr/time-programming/#\\_scope](http://www.catb.org/esr/time-programming/#_scope). Acesso em: 24 dez. 2019.

## Próxima aula

- Mais detalhes de bibliotecas gráficas em C com a API OpenGL;
- Captura de eventos – teclado e mouse;
- Entrada e saída por console e arquivo.

## Explore mais

O material a seguir é útil para explorar mais sobre bibliotecas em C para sistemas operacionais diferentes e outros compiladores:

- [Biblioteca C](#).
- [Aula 11 – Bibliotecas de função](#).

- [Como criar arquivos de cabeçalho em linguagem C.](#)