



# Bibliotecas e APIs para Linguagem C

Você vai estudar os fundamentos da programação em C, o uso de bibliotecas para manipulação de datas, horários e gráficos e a implementação de interfaces via OpenGL. Você aprenderá sobre estruturas de dados, controle de fluxo e interação com APIs gráficas, adquirindo habilidades para desenvolver softwares robustos e enfrentando desafios no desenvolvimento de sistemas e aplicações interativas.

Prof. Marcos Santana Farias

## Preparação

Para iniciar seus estudos sobre o assunto, é recomendado instalar o DevC++ na versão 5.11. O tutorial de instalação está anexado [aqui](#).

## Objetivos

- Descrever os conceitos básicos da linguagem C e a sua importância para a programação de sistemas.
- Aplicar bibliotecas de rotinas gráficas e calendário para exemplificar a modularização de aplicações de software básico.
- Aplicar a API OpenGL e a biblioteca GLUT para realizar gráficos e desenhos no monitor do computador.
- Criar aplicações que percebam as atuações do usuário em dispositivos de entrada, como o teclado e o mouse.

## Bibliotecas e APIs para linguagem C

Neste vídeo, exploraremos os pilares da programação em C, desde a estrutura básica de programas até conceitos como alocação dinâmica e controle de fluxo. Confira!



### Conteúdo interativo

Acesse a versão digital para assistir ao vídeo.

# Estrutura de um programa em C

Explorar a estrutura de um programa em C oferece uma visão clara sobre como a linguagem C funciona, desde a inclusão de bibliotecas até a execução de comandos e a definição de variáveis e constantes. Compreender isso fornece as condições para criar programas eficazes que operam de maneira precisa e eficiente em diversos contextos de desenvolvimento de software.

Para iniciar seus estudos sobre o assunto, confira no vídeo a seguir o processo de compilação e o uso de diretivas de compilação. Serão utilizados exemplos práticos para ilustrar como esses elementos se combinam para criar programas funcionais e eficientes.



### Conteúdo interativo

Acesse a versão digital para assistir ao vídeo.

Começar a programar em C é como montar um pequeno robô para fazer tarefas específicas para você. Cada parte do programa tem sua função, e entender cada uma delas, bem como sua estrutura, é o que fará seu robô funcionar corretamente.

## Estrutura básica de um programa em C

Imagine que você está escrevendo uma receita de bolo, na qual há uma introdução, a lista de ingredientes e o passo a passo. Em C, é parecido! Veja:

- **Introdução:** você inclui bibliotecas com funções prontas, que ajudam em tarefas comuns, como entrada e saída de dados. É como se você pegasse utensílios e ingredientes prontos para usar.

c

```
#include // inclui definições para funções de entrada e saída, que permitem realizar operações como ler dados do teclado e exibir informações na tela do computador.
```

- **Função principal (int main()):** você coloca as instruções que o computador vai seguir, ou seja, a receita em si.

c

```
int main() {  
    printf("Olá, mundo!");  
    return 0;  
}
```

## Processo de compilação

Transforma o código-fonte em C em código de máquina executável pelo sistema operacional. O compilador verifica a sintaxe do código e o converte em um programa executável. Esse processo inclui várias etapas:

- Pré-processamento: executa diretivas e remove comentários.
- Compilação: converte para linguagem de montagem ou código objeto.
- Montagem: transforma em código objeto binário.
- Link-edição: combina módulos de código e bibliotecas em um arquivo executável.

## Variáveis e constantes

Vamos continuar utilizando a metáfora da receita de bolo para entendermos os importantes conceitos da estrutura de um programa em C. As variáveis são como os recipientes em que você pode armazenar os ingredientes, conforme a receita pede. Por exemplo, se você precisa ajustar a quantidade de açúcar conforme o seu paladar, isso seria uma variável.

```
c
int idade = 30; // Armazenando a idade de alguém em uma variável
```

Constantes, por outro lado, são como os ingredientes que não mudam, por exemplo, a quantidade de ovos em uma receita específica. Dessa forma, uma vez definidos, você não pode alterá-los.

```
c
const float PI = 3.14; // PI sempre terá esse valor, não muda
```

## Diretivas de compilação

São instruções especiais no início do código, que dizem ao compilador para incluir certos recursos ou comportar-se de determinada maneira durante a compilação do programa. São como instruções especiais para quem vai preparar o bolo. Por exemplo, pré-aquecer o forno.

```
c
#define MAXIMO 100
```

O código define MAXIMO como 100, e sempre que você usar MAXIMO no seu código, o compilador o substituirá por 100.

## Atividade 1

### Questão 1

João está desenvolvendo um programa em C que calcula a área de um círculo. Ele precisa utilizar o valor de PI e garantir que ele não seja alterado acidentalmente em seu código. Além disso, ele quer definir um valor máximo para o raio do círculo que pode ser calculado. Com base nos conceitos de variáveis, constantes e diretivas de compilação em C, qual das seguintes opções João deveria implementar corretamente em seu programa?

A

```
int PI = 3.14; #define RAIO_MAX 100
```

B

```
const float PI = 3.14; #define RAIO_MAX 100
```

C

```
#define PI 3.14; int RAIO_MAX = 100
```

D

```
float PI = 3.14; const int RAIO_MAX = 100
```

E

```
#define PI 3.14; #define RAIO_MAX 100
```



A alternativa B está correta.

Constante PI: a declaração `const float PI = 3.14;` assegura que o valor de PI seja tratado como uma constante, ou seja, um valor que não pode ser alterado depois de inicializado.

Diretiva RAIO\_MAX: `#define RAIO_MAX 100` define valor máximo para o raio que não precisa ser alterado e pode ser facilmente acessado em todo o código.

As outras opções apresentam problemas, Confira:

- `int PI = 3.14;` permite que PI seja alterado, o que não é desejável.
- `#define PI 3.14;` trata PI como uma macro, o que é menos seguro e pode levar a problemas de precisão devido à falta de tipo.
- `float PI = 3.14;` também permite que PI seja alterado, e `const int RAIO_MAX = 100;` usa `const` quando `#define` seria mais adequado para diretivas de compilação.

## Tipos básicos de dados, classes de armazenamento, especificadores e operadores

A linguagem C é conhecida por sua potência e precisão, sendo utilizada em uma variedade de aplicações, que vão desde sistemas operacionais até softwares de alto desempenho. Entre seus elementos essenciais, destacam-se os tipos básicos de dados, como `int`, `char` e `float`, que definem a natureza das variáveis.

As classes de armazenamento, como `static` e `extern`, controlam o tempo de vida e a visibilidade das variáveis. Por último, os especificadores, como `signed` e `unsigned`, refinam a definição de tipos de dados, enquanto os operadores, incluindo aritméticos e lógicos, permitem manipulações diversas em expressões e variáveis.

Assista, neste vídeo, aos fundamentos da linguagem C, abordando tipos de dados, classes de armazenamento, especificadores, operadores e sua precedência.



### Conteúdo interativo

Acesse a versão digital para assistir ao vídeo.

## Tipos básicos de dados

São fundamentais em C. Os mais comuns incluem:

### Inteiros (int)

São usados para guardar números sem casas decimais, como a contagem de usuários logados em um sistema.

### Ponto flutuante (float, double)

São empregados em números com casas decimais e úteis em cálculos de precisão, como taxas de juros.

### Caracteres (char)

São importantes em processamento de texto, pois armazenam um único caractere.

## Classes de armazenamento

Definem a vida útil e a visibilidade de variáveis ou funções. Confira mais detalhes a seguir!

### auto

É usada para variáveis locais e desaparece após o uso em um bloco ou função.

### static

Preserva seu valor entre chamadas de função e é ideal para manter estados, como um contador de visitas.

### extern

Informa que a variável é definida externamente e é comum em programas com múltiplos arquivos.

### register

Recomenda que a variável seja armazenada em um registro da CPU para acesso rápido, especialmente em loops frequentes.

## Especificadores ou modificadores de tipo

Ajustam o alcance e o tamanho dos dados. Vamos conferir mais detalhes a seguir!

### unsigned int

Guarda apenas valores positivos, maior do que um int comum. É ideal para contagens.

### long

Aumenta a capacidade de armazenamento. É adequado para números muito grandes.

## Operadores e precedência

São essenciais para realizar operações matemáticas e lógicas. Eles incluem aritméticos (+, -, \*, /), lógicos (&&, ||, !) e de comparação (==, !=, >, <). Já a precedência dos operadores define a ordem de avaliação das operações, na qual operadores de maior precedência são avaliados primeiro.

Além disso, parênteses podem ser usados para alterar a ordem de avaliação. Por exemplo, para priorizar a adição em  $(a + b) * c$ , os parênteses garantem que  $a$  e  $b$  são somados antes da multiplicação. Expressões combinam valores, variáveis e operadores para produzir novo valor com operadores de atribuição, como  $=$ ,  $+=$ , e  $*=$ , que simplificam o código ao combinar aritmética com atribuição.

## Atividade 2

### Questão 1

Para garantir que o total de produtos em estoque seja mantido corretamente após cada transação, você decide usar em seu programa em C a variável `static int totalProdutos;` para manter o registro atualizado entre chamadas de função. Com base nessa informação, analise as afirmativas a seguir e, depois, assinale a alternativa correta.

Asserção (A): o uso de uma variável `static` dentro de uma função permite que seu valor seja preservado entre diferentes chamadas da função.

Razão (R): uma variável `static` pode ajudar a otimizar o código, pois o compilador consegue alocá-la em um local de memória de acesso mais rápido.

A

A e R são verdadeiras, e R é uma justificativa correta de A.

B

A e R são verdadeiras, mas R não é uma justificativa correta de A.

C

A é verdadeira, mas R é falsa.

D

A é falsa, mas R é verdadeira.

E

A e R são falsas.



A alternativa C está correta.

Variáveis declaradas como `static` dentro de funções retêm seu valor entre chamadas, o que é útil em aplicações (como gerenciamento de estoque) para manter totais atualizados continuamente. No entanto, a afirmação de que variáveis `static` otimizam o código por serem alocadas mais rapidamente em memória é incorreta, pois elas são alocadas na área de dados do programa, e a otimização não se deve à velocidade de acesso à memória.

## Estruturas condicionais e de repetição

São as ferramentas que permitem controlar o fluxo de um programa e repetir tarefas de maneira eficiente. Esse conhecimento teórico facilita a resolução de problemas práticos e serve como base para explorar conceitos mais avançados de programação.

Veja, neste vídeo, as estruturas condicionais e de repetição em C, demonstrando como `if`, `else`, `switch-case`, `for`, `while` e `do-while` podem ser usadas para controlar o fluxo de programas e automatizar tarefas repetitivas.



Conteúdo interativo

Acesse a versão digital para assistir ao vídeo.

As estruturas condicionais são fundamentais na programação, porque permitem que o código tome decisões que se baseiam em condições específicas. Vamos conhecer mais sobre elas a seguir!

### Estruturas condicionais

São como as decisões que tomamos todos os dias. Por exemplo, se estiver chovendo, você pega um guarda-chuva, caso contrário, você sai sem ele. Em C, isso é feito principalmente por meio das instruções **`if`**, **`else`** e **`else if`**.

### A estrutura `if`

A instrução **`if`** é a mais básica das estruturas condicionais e permite que você execute um bloco de código somente se uma condição específica for verdadeira. A sintaxe básica do **`if`** é a seguinte:

```
c
if (condição) {
    // Bloco de código que é executado se a condição é verdadeira
}
```

Exemplo:



```
c
#include
int main() {
    int idade = 18;

    if (idade >= 18) {
        printf("Você é maior de idade!\n");
    }
    return 0;
}
```

No exemplo, se a idade for 18 ou mais, o programa imprimirá **Você é maior de idade!**

## A estrutura else

Frequentemente, queremos ter um comportamento alternativo caso a condição do **if** não seja verdadeira. É aqui que o **else** entra.

```
c
if (condição) {
    // Bloco de código que é executado se a condição é verdadeira
} else {
    // Bloco de código que é executado se a condição é falsa
}
```

## A estrutura if-else

Você pode verificar múltiplas condições sequencialmente. Isso é feito usando uma combinação de **if**, **else if** e **else**.

```
c
if (condição1) {
    // Executa se condição1 é verdadeira
} else if (condição2) {
    // Executa se condição1 é falsa e condição2 é verdadeira
} else {
    // Executa se todas as condições anteriores são falsas
}
```

Lembre-se: o uso de múltiplos **else if** pode tornar o código menos legível, especialmente com muitas condições. Em tais casos, pode ser melhor usar a estrutura que veremos a seguir.

## A estrutura switch-case

Funciona de maneira similar a uma série de **if-else**, mas de forma mais limpa e organizada, especialmente quando lidamos com muitas escolhas diretas e baseadas em uma única variável.

```
c
switch (expressão) {
    case valor1:
        // Bloco de código para valor1
        break;
    case valor2:
        // Bloco de código para valor2
        break;
    default:
        // Bloco de código se nenhum caso anterior corresponder
}
```

Vamos imaginar que você está desenvolvendo um software para uma máquina de venda automática. O usuário pode pressionar botões numerados para escolher um produto. Veja como você poderia usar um **switch-case** para implementar essa opção:

```
c
#include

int main() {
    int escolha;

    printf("Escolha seu produto: 1 para Água, 2 para Refrigerante, 3 para Suco\n");
    scanf("%d", &escolha);

    switch (escolha) {
        case 1:
            printf("Você escolheu Água.\n");
            break;
        case 2:
            printf("Você escolheu Refrigerante.\n");
            break;
        case 3:
            printf("Você escolheu Suco.\n");
            break;
        default:
            printf("Opção inválida. Por favor, escolha um produto válido.\n");
    }

    return 0;
}
```

## Estruturas de repetição

Permitem que você execute um bloco de código várias vezes. Em C, as principais estruturas de repetição são **for**, **while** e **do-while**.

### Laço for

É tipicamente usado quando o número de iterações é conhecido antes de entrar no laço. Ele é muito utilizado para iterar sobre estruturas de dados ou para realizar operações repetidas em uma quantidade específica de vezes. A sintaxe básica é:

```
c
for (inicialização; condição; incremento) {
    // Código a ser repetido
}
```

Confira alguns detalhes importantes sobre o laço for e que podem ser vistos em nosso exemplo:

**Inicialização:** usada, geralmente, para definir o contador do loop.

**Condição:** enquanto for verdadeira, o loop continuará a executar.

**Incremento:** atualiza o contador do loop após cada iteração.

Observe o próximo exemplo:

```
c
#include

int main() {
    for (int i = 0; i < 5; i++) {
        printf("Número %d\n", i);
    }
    return 0;
}
```

Podemos ver no exemplo dado que o laço for imprime números de 0 a 4. O loop executa cinco vezes, conforme definido pela condição.

## Laço while

É usado quando o número de iterações não é necessariamente conhecido antes de iniciar o loop.

```
c
while (condição) {
    // Código a ser repetido
}
```

## Laço do-while

É uma variação do **while**, garantindo que o bloco de código seja executado, pelo menos, uma vez, pois a condição é verificada após a execução do bloco.

```
c
do {
    // Código a ser repetido
} while (condição);
```

## Atividade 3

Você está desenvolvendo um software para um caixa eletrônico. O sistema deve perguntar repetidamente ao usuário se deseja continuar realizando saques até que o usuário decida parar. Qual das seguintes opções de código em C implementa corretamente essa funcionalidade?

```
c
#include
int main() {
    char continuar;
    do {
        printf("Saque realizado. Deseja fazer outro saque? (s/n): ");
        scanf(" %c", &continuar);
        // Código para realizar o saque omitido
    } while (condição);
    return 0;
}
```

A

```
} while (continuar == 's' || continuar == 'S')
```

B

```
} while (continuar == 'n' || continuar == 'N')
```

C

```
} while (continuar != 'n' && continuar != 'N')
```

D

```
} while (continuar != 's' && continuar != 'S')
```

E

```
} while (continuar == 'y' || continuar == 'Y')
```



A alternativa A está correta.

A condição `(continuar == 's' || continuar == 'S')` garante que o loop prossiga apenas se o usuário explicitamente indicar que deseja fazer outro saque, inserindo 's' ou 'S'. As demais alternativas ou invertem a lógica necessária (B e C), usam uma condição que finalizaria o loop incorretamente (se o usuário quiser continuar (D)) ou utilizam uma resposta ('y' ou 'Y') que não foi especificada como válida no contexto do problema (E).

## Vetores, matrizes, ponteiros e alocação dinâmica de memória

Em programação, especialmente em C, certos conceitos — como vetores, matrizes, ponteiros e alocação dinâmica de memória — formam a base para entender como os programas armazenam, acessam e manipulam dados de maneira eficiente. Compreender esses conceitos ajuda a escrever códigos melhores e prepara você

para enfrentar problemas complexos no mundo real, como desenvolvimento de sistemas, jogos e aplicações que exigem desempenho otimizado.

Iniciando seus estudos, neste vídeo, vamos abordar conceitos básicos de vetores, matrizes, ponteiros e alocação dinâmica de memória em C, o quais são fundamentais para otimizar e tornar a programação mais eficaz. Demonstraremos com exemplos práticos como essas estruturas funcionam.



#### Conteúdo interativo

Acesse a versão digital para assistir ao vídeo.

## Vetores e matrizes

Vamos começar entendendo o que são vetores em C: são sequências de elementos do mesmo tipo e acessados por índices. Eles são semelhantes aos vagões numerados de um trem de carga. Por exemplo, `int numeros[10]`; define um vetor de 10 inteiros.

Matrizes são estruturas bidimensionais, comparáveis a prédios de apartamentos, em que cada andar é um vetor, e cada apartamento é um elemento. Elas são úteis para organizar dados em tabelas, como `int tabela[4][5]`; que define uma matriz de quatro linhas por cinco colunas.

## Ponteiros

São variáveis que armazenam endereços de memória, funcionando como coordenadas que direcionam para locais específicos na memória do computador. Eles são importantes para a manipulação direta de dados na memória, aumentando a eficiência. A declaração de um ponteiro — como `int *ponteiroParaInteiro`; — usa o operador `*` para indicar que a variável aponta para um inteiro.

Usar ponteiros permite eficiência de memória, manipulação de dados em nível baixo e construção de estruturas complexas de dados. Eles também são essenciais para passar matrizes e strings para funções eficientemente.

## Alocação dinâmica de memória

Realizada por meio das funções `malloc`, `calloc`, e `free`, a alocação dinâmica de memória permite gerenciar a memória de forma flexível, alocando e liberando espaço conforme necessário. Isso é particularmente útil em aplicativos que lidam com quantidades variáveis de dados de entrada, permitindo que o uso de memória seja ajustado em tempo real.

Por exemplo, `int *numeros = malloc(10 * sizeof(int))`; aloca espaço para 10 inteiros, e `free(numeros)`; libera a memória quando não é mais necessária, otimizando o uso de recursos e melhorando o desempenho do aplicativo.

## Atividade 4

### Questão 1

Você está projetando um programa em C para automatizar a emissão de bilhetes em uma estação de metrô. O sistema deve imprimir bilhetes com base na idade do passageiro, oferecendo bilhetes comuns para adultos e com desconto para crianças e idosos. Considere as afirmações a seguir sobre uma solução para o problema e a relação entre elas.

Asserção (A): para implementar a diferenciação de bilhetes no sistema da estação de metrô, é necessário utilizar uma estrutura condicional if-else.

Razão (R): a estrutura condicional if-else permite executar diferentes blocos de código com base em condições específicas, como a faixa etária de um passageiro.

Indique a alternativa correta.

A

Tanto a asserção como a razão são verdadeiras, e a razão é uma explicação correta da asserção.

B

Tanto a asserção como a razão são verdadeiras, mas a razão não é uma explicação correta da asserção.

C

A asserção é verdadeira, mas a razão é falsa.

D

A asserção é falsa, mas a razão é verdadeira.

E

Tanto a asserção como a razão são falsas.



A alternativa A está correta.

Para implementar a diferenciação de bilhetes no sistema da estação de metrô, realmente é necessário utilizar uma estrutura condicional if-else. Ela permite que o programa tome decisões diferentes com base na idade do passageiro, emitindo bilhetes comuns para adultos e outros com desconto para crianças e idosos.

A razão é verdadeira porque a estrutura condicional if-else, de fato, permite executar diferentes blocos de código com base em condições específicas, como a faixa etária de um passageiro. Logo, (R) explica por que (A) é verdadeira.

## Codificando um programa básico em C

Nesta prática de programação em C, exploraremos alocação dinâmica, ponteiros e estruturas de controle, como condicionais e laços. Aprenderemos a gerenciar memória com malloc e free, utilizar ponteiros para manipular dados e implementar lógicas com if e for, visando à eficiência e ao código limpo.

Assista ao vídeo a seguir, no qual demonstraremos um tutorial com o passo a passo sobre programação em C utilizando a IDE on-line GDB. Assim, compreenderemos como escrever e executar um programa que utiliza

alocação dinâmica, ponteiros e estruturas de controle para gerenciar uma lista de números inteiros, incluindo como inserir dados, exibir resultados e gerenciar a memória de forma eficaz.



#### Conteúdo interativo

Acesse a versão digital para assistir ao vídeo.

## Roteiro de prática

Objetivo: criar um programa em C que utilize alocação dinâmica, ponteiros e estruturas condicionais e de repetição para gerenciar uma lista de números inteiros.

### Passo 1: configuração inicial na IDE GDB on-line

- Acesse a IDE GDB on-line em: [https://www.onlinegdb.com/online\\_c\\_compiler](https://www.onlinegdb.com/online_c_compiler).
- Selecione a linguagem C no menu dropdown.

### Passo 2: escrever o código

Vamos escrever um programa que permita ao usuário inserir uma quantidade definida de números, armazená-los em um array dinâmico e depois exibir os números inseridos.

```

c

#include
#include

int main() {
    int *array;
    int size, i;

    printf("Quantos números você deseja inserir? ");
    scanf("%d", &size);

    // Alocação dinâmica de memória
    array = (int*) malloc(size * sizeof(int));

    // Verificação de alocação bem-sucedida
    if (array == NULL) {
        printf("Erro de alocação de memória.\n");
        return -1; // Encerra o programa com erro
    }

    // Inserção de números no array
    for (i = 0; i < size; i++) {
        printf("Digite o número %d: ", i + 1);
        scanf("%d", &array[i]);
    }

    // Exibição dos números
    printf("Você inseriu: ");
    for (i = 0; i < size; i++) {
        printf("%d ", array[i]);
    }

    // Liberação da memória alocada
    free(array);

    return 0;
}

```

### Passo 3: executar o código

- Clique no botão Run na IDE para executar o programa.
- Insira os valores conforme solicitado pelo prompt do programa.

## Faça você mesmo!

#### Questão 1

Números primos são números inteiros maiores que 1 e têm apenas dois divisores distintos: 1 e eles mesmos. Escreva um programa em C que descubra qual seria o 100º número primo.

Chave de resposta

**Código:**

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



```
int main() {  
  
    int count = 0, num = 2, i;  
    int N = 100; // Deseja-se encontrar o 100º número primo  
    int isPrime;  
  
    while (count < N) {  
        isPrime = 1; // Assume que o número é primo  
        for (i = 2; i * i <= num; i++) {  
            if (num % i == 0) {  
                isPrime = 0; // O número não é primo  
                break;  
            }  
        }  
        if (isPrime) {  
            count++;  
            if (count == N) {  
                printf("%dº número primo é %d.\n", N, num);  
            }  
        }  
        num++;  
    }  
    return 0;  
}
```

#### Resultado do código:

O 100º número primo é 541.

#### Explicação do código:

- **Variáveis:** count conta primos, num é o número testado, isPrime indica primalidade (1 para primo).
- **Laço externo:** continua até count atingir 100, indicando que o 100º primo foi encontrado.
- **Laço interno:** for (i = 2; i \* i <= num; i++) verifica divisores até a raiz quadrada de num, reduzindo as verificações.
- **Atualização de contadores:** se isPrime é 1 após verificar, count incrementa e imprime o 100º primo encontrado.

# Bibliotecas para datas e horários

Em C, funções são blocos de código que executam tarefas específicas, tornando o programa mais modular e reutilizável. Elas têm tipo de retorno, nome, parâmetros e um corpo. Bibliotecas em C são conjuntos de funções pré-compiladas que oferecem funcionalidades comuns, como manipulação de strings e operações matemáticas, facilitando o desenvolvimento.

Neste vídeo, exploraremos as funções e bibliotecas em C, focando na biblioteca padrão do C. Detalharemos a `time.h`, que lida com data e hora. Além disso, mostraremos como essas bibliotecas facilitam tarefas comuns como leitura de arquivos e manipulação de datas.



### Conteúdo interativo

Acesse a versão digital para assistir ao vídeo.

As bibliotecas em C podem ser padrões, incluídas no compilador, ou de terceiros, isto é, criadas por outros programadores. As bibliotecas padrão em C são aquelas que vêm incluídas com o compilador e integram a linguagem. Elas fornecem funções básicas e essenciais que cobrem muitas necessidades de programação.

Para utilizar as bibliotecas, os programadores geralmente incluem os cabeçalhos apropriados no início de seus arquivos fonte com a diretiva `#include`.

## Biblioteca padrão do C

É uma coleção essencial de funções predefinidas, macros e tipos de dados, especificada por padrões como ANSI C e ISO C. Um componente-chave é a biblioteca `stdio.h`, responsável por operações de entrada e saída. O termo `stdio` deriva de `standard input and output`. Essa biblioteca oferece funções para leitura e escrita de dados, incluindo as conhecidas `printf()` e `scanf()`.

## Biblioteca padrão do C para data e hora

É uma seção governada pelo cabeçalho `time.h`, que disponibiliza funções e tipos para manipulação de datas e horas, essenciais para medir tempos de execução e trabalhar com o sistema de datas e horas.

## Principais tipos de dados em `time.h`

- `time_t`: representa o tempo em segundos desde o Epoch (1 de janeiro de 1970).
- `struct tm`: armazena componentes de tempo de forma quebrada, como ano, mês e dia, entre outros.

## Funções importantes em `time.h`

São funcionalidades essenciais para aplicativos que dependem de temporização ou trabalham com datas programaticamente. Confira mais detalhes a seguir!

- `time()`: obtém o tempo atual do sistema.
- `gmtime()` e `localtime()`: convertem `time_t` para `struct tm` em UTC ou horário local.
- `mktime()`: converte `struct tm` de volta para `time_t`.
- `strftime()`: formata data e hora em `struct tm` para uma string.
- `difftime()`: calcula a diferença em segundos entre dois tempos (`time_t`).

- `asctime()` e `ctime()`: convertem tempo para uma string legível.

## Atividade 1

### Questão 1

A linguagem de programação C é amplamente utilizada devido à sua eficiência e flexibilidade. Parte da sua robustez vem das bibliotecas padrão que acompanham a linguagem, fornecendo funcionalidades essenciais para diversas operações. Duas dessas bibliotecas padrão são `<stdio.h>` e `<time.h>`. Considere as seguintes afirmações sobre programação em C, especificamente sobre o uso das bibliotecas padrão `stdio.h` e `time.h`:

I. A função `printf()` encontrada na biblioteca `stdio.h` é usada para imprimir dados na saída padrão.

II. `time_t` é um tipo definido na biblioteca `time.h` que representa o tempo em milissegundos desde o Unix Epoch.

III. `localtime()` é uma função da `time.h` que converte um valor `time_t` para a hora local, representada por uma estrutura `struct tm`.

Selecione a seguir a alternativa correta.

A

Apenas I e II estão corretas.

B

Apenas II e III estão corretas.

C

Apenas I e III estão corretas.

D

Apenas I está correta.

E

Apenas II está correta.



A alternativa C está correta.

A função `printf()` é amplamente utilizada para imprimir dados formatados na saída padrão (geralmente o console). Ela é uma das funções mais comuns e fundamentais da biblioteca `<stdio.h>`, permitindo a exibição de texto e variáveis em diferentes formatos.

O tipo `time_t` realmente é definido na biblioteca `<time.h>`, mas ele representa o tempo em segundos (e não em milissegundos) desde o Unix Epoch (00:00:00 UTC em 1 de janeiro de 1970). A definição precisa do tipo `time_t` pode variar entre implementações, mas, por padrão, ele conta o tempo em segundos. Por isso a afirmação II está incorreta.

A função `localtime()` é usada para converter um valor do tipo `time_t` (que representa o tempo em segundos desde o Unix Epoch) em uma estrutura `struct tm`, que contém a data e a hora local. Essa função ajusta o tempo de acordo com o fuso horário local e o horário de verão, se aplicável.

## Biblioteca para gráficos

O padrão ANSI C não inclui rotinas gráficas em sua biblioteca padrão. Entretanto, as bibliotecas para gráficos são essenciais no desenvolvimento de jogos, simulações e interfaces, sendo fundamentais ao ensino de programação gráfica.

Neste vídeo, exploraremos a importância das bibliotecas gráficas na programação em C, destacando as funcionalidades e as aplicações de soluções como OpenGL, SDL, Cairo e `graphics.h`. Daremos especial ênfase à utilidade educacional da `graphics.h` para introduzir conceitos básicos de gráficos computacionais.



### Conteúdo interativo

Acesse a versão digital para assistir ao vídeo.

O padrão C ANSI recorre a soluções como OpenGL e SDL porque não inclui rotinas gráficas. Além disso, embora ultrapassada, a biblioteca `graphics.h` continua relevante para ensinar programação gráfica em C, introduzindo conceitos básicos de gráficos computacionais de forma simples e direta, auxiliando o processo ensino-aprendizagem.

```
c
#include

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();
}
```

O uso da biblioteca `graphics.h` em C permite acesso à funções gráficas por meio de:

**#include**

Integra a biblioteca ao programa.

**initwindow(800, 800)**

Cria uma janela de 800×800 pixels para renderização gráfica.

**Definição de variáveis**

- Possui as coordenadas left, top, right, bottom para um retângulo.
- Tem x, y para o centro de um círculo, e radius para o raio.

**rectangle(left, top, right, bottom)**

Desenha um retângulo.

**circle(x, y, radius)**

Desenha um círculo.

**bar(left + 300, top, right + 300, bottom)**

Desenha um retângulo preenchido, deslocado 300 pixels à direita.

**line(left - 10, top + 150, left + 410, top + 150)**

Desenha uma linha horizontal estendendo-se do retângulo.

**ellipse(x, y + 200, 0, 360, 100, 50)**

Desenha uma elipse completa com raios horizontal e vertical.

**outtextxy(left + 100, top + 325, "Meu programa grafico")**

Exibe texto nas coordenadas definidas.

**getch()**

Mantém a janela gráfica aberta até uma tecla ser pressionada.

Outras bibliotecas gráficas em C incluem:

1

SDL

Útil para gráficos e áudio. É apropriado em jogos.

2

Allegro e SFML

Válido para multimídia e interfaces gráficas.

3

Cairo

Apropriado para gráficos vetoriais precisos.

4

Plplot

Aplicável para gráficos científicos.

5

GTK

Útil para interfaces gráficas no Linux.

Essas bibliotecas expandem a capacidade gráfica em C, cada uma adequada para diferentes tipos de aplicações gráficas.

## Atividade 2

### Questão 1

Enquanto bibliotecas modernas, como OpenGL e SDL, são frequentemente usadas para aplicações avançadas e jogos, a biblioteca `graphics.h`, embora antiga, ainda é utilizada em contextos educacionais para ensinar conceitos básicos de gráficos em C. Com base nessa afirmativa, analise as informações a seguir e assinale a alternativa correta.

Asserção (A): a biblioteca `graphics.h` é frequentemente utilizada em contextos educacionais para introduzir conceitos de programação gráfica em C.

Razão (R): a biblioteca `graphics.h` é parte do antigo Borland's Turbo C e do compilador Borland C++.

A

A e R são verdadeiras, e R é a justificativa correta de A.

B

A e R são verdadeiras, mas R não é a justificativa correta de A.

C

A é verdadeira, mas R é falsa.

D

A é falsa, mas R é verdadeira.

E

A e R são falsas.



A alternativa B está correta.

A biblioteca graphics.h é amplamente usada em ambientes educacionais para ensinar conceitos básicos de programação gráfica. Embora seja antiga e limitada em comparação com bibliotecas mais modernas, como OpenGL e SDL, sua simplicidade faz com que seja adequada para iniciantes. Por isso (A) está correta.

A biblioteca graphics.h faz parte do antigo Borland's Turbo C e do compilador Borland C++. Essas ferramentas foram populares em ambientes educacionais nas décadas passadas, e a biblioteca graphics.h era frequentemente utilizada para ensinar gráficos em C.

Embora ambas as afirmações sejam verdadeiras, (R) não justifica diretamente (A). A Razão explica a origem da biblioteca graphics.h, mas não por que ela é usada em contextos educacionais. A verdadeira justificativa para o uso da graphics.h em educação é sua simplicidade e facilidade de uso, que descomplica a introdução de conceitos básicos de gráficos a estudantes, e não apenas o fato de ela fazer parte do Turbo C e do Borland C++.

## Principais APIs para C

Ao explorar a programação gráfica em C, é preciso entender as diferenças e as aplicações de bibliotecas e APIs gráficas. Esse conhecimento fundamenta o desenvolvimento eficiente de software, permitindo aos programadores escolher a ferramenta certa para tarefas específicas, desde renderização de imagens até interação complexa com hardware gráfico, maximizando a eficácia e a portabilidade dos projetos.

Acompanhe, neste vídeo, as comparações das características e dos usos específicos das APIs DirectX e OpenGL, enfatizando suas aplicações em diferentes plataformas e cenários de desenvolvimento de jogos e aplicações gráficas. Confira!



Conteúdo interativo

Acesse a versão digital para assistir ao vídeo.

Bibliotecas e APIs gráficas desempenham papéis distintos e complementares. Esse entendimento é fundamental para escolher a abordagem correta ao desenvolver aplicações gráficas, sejam elas simples ou complexas.

## Diferença entre bibliotecas e APIs gráficas

Ao criar gráficos em C, você pode utilizar bibliotecas e APIs gráficas, pois cada uma possui estruturas e propósitos distintos.

Bibliotecas gráficas são coleções de funções e procedimentos prontos que encapsulam operações comuns, permitindo que os desenvolvedores não tenham que reescrever um código frequentemente. Elas oferecem funcionalidades como renderização de imagens e manipulação de texturas, exemplificadas por SDL, Allegro e Cairo.

APIs gráficas (ou *application programming interface*) são conjuntos de regras e especificações que definem como os programas podem interagir com software ou hardware. No contexto gráfico, uma API estabelece métodos para tarefas como renderização 3D e manipulação de sombreadores. APIs como OpenGL e DirectX fornecem camadas de abstração que permitem desenvolver recursos gráficos avançados sem preocupações com os detalhes do hardware.

A biblioteca é um conjunto de funcionalidades prontas para uso. A API oferece uma estrutura para desenvolver e manipular componentes gráficos, podendo ser suportada por várias bibliotecas.

### API DirectX

É uma série de APIs da Microsoft para facilitar o desenvolvimento de jogos e aplicativos multimídia em Windows. Ela inclui funções para renderização gráfica 2D e 3D, áudio e gerenciamento de dispositivos de entrada. Sua componente, Direct3D, interage diretamente com a GPU para renderização rápida de gráficos 3D complexos.

DirectX é essencial para jogos em Windows e consoles Xbox, diferenciando-se da OpenGL pela sua otimização para o ambiente Windows e sua eficiência na interação com o hardware.

## Atividade 3

### Questão 1

Ao desenvolver jogos para plataformas Windows, um programador deve escolher entre várias APIs gráficas. Qual API é altamente recomendada para esse propósito, pois oferece interação direta com o hardware de vídeo e áudio, otimizando a execução de jogos e aplicações multimídia, e é especialmente eficiente devido à sua capacidade de manipular diretamente a GPU?

A

OpenGL

B

SDL

C

DirectX

D



Allegro

E

Cairo



A alternativa C está correta.

DirectX é altamente recomendada para o desenvolvimento de jogos em plataformas Windows. Essa escolha se deve à sua capacidade de interação direta com o hardware de vídeo e áudio, o que otimiza significativamente a execução de jogos e aplicações multimídia. Ela é projetada para ser especialmente eficiente no ambiente Windows, aproveitando ao máximo os recursos do sistema operacional e da GPU. Suas funcionalidades abrangem renderização gráfica 2D e 3D, áudio e gerenciamento de dispositivos de entrada, tornando-a uma solução completa e integrada para desenvolvedores de jogos.

## Bibliotecas básicas para jogos

As bibliotecas em C são fundamentais no desenvolvimento de jogos, oferecendo controle direto sobre recursos do sistema e permitindo otimizações específicas para melhorar o desempenho. Entender essas ferramentas permite a criação de jogos eficientes e responsivos em diversas plataformas.

Confira, neste vídeo, as principais bibliotecas, como SDL, Allegro, SFML e Raylib, e de que modo cada uma contribui para criar jogos eficientes e interativos em diferentes plataformas.



### Conteúdo interativo

Acesse a versão digital para assistir ao vídeo.

As bibliotecas em C são essenciais no desenvolvimento de jogos, dado o alto desempenho e a eficiência da linguagem. Ela permite controle preciso sobre a memória e outros recursos do sistema, fatores importantes para jogos que demandam resposta rápida e intenso processamento. C oferece otimizações que aprimoram significativamente o desempenho dos jogos.

A alta portabilidade de C facilita a adaptação de jogos para diversas plataformas, enquanto a maturidade da linguagem assegura um ecossistema desenvolvido de bibliotecas e ferramentas, além de uma comunidade ativa de suporte e aprendizado. Tais características tornam C uma escolha popular entre desenvolvedores de jogos, especialmente aqueles que visam desempenho superior e otimização rigorosa.

## Características de uma biblioteca básica para jogos

Confira, a seguir, o que uma biblioteca essencial para desenvolvimento de jogos deve abranger:

### Sistema de janelas

Criação e gerenciamento de janelas, com funcionalidades como ajuste de tamanho e resposta a eventos.

### Gráficos e renderização

Ferramentas para desenhar imagens, formas e texturas. Está frequentemente integradas a APIs, como OpenGL ou DirectX, para suportar gráficos 2D e 3D.

### Manipulação de eventos

Resposta a interações do usuário por meio de dispositivos, como teclado, mouse e *joystick*.

### Áudio

Funcionalidades para reprodução de sons e músicas.

### Rede

Comunicação em rede é essencial para jogos multiplayer.

### Temporização e controle de frames

Gerenciamento da taxa de atualização para a fluidez do jogo.

### Sistema de arquivos

Leitura e escrita de arquivos para carregar recursos ou salvar estados de jogo.

### Matemática e física

Bibliotecas para cálculos geométricos e físicos necessários para movimentação e interações.

## Principais bibliotecas para jogos em C

Selecionamos as bibliotecas a seguir porque permitem a criação de jogos dinâmicos e interativos em C, sendo adaptáveis a múltiplas plataformas. Veja!

1

#### SDL (simple directmedia layer)

Possui acesso de baixo nível a sistemas de vídeo, áudio e dispositivos de entrada, com suporte para gráficos 3D via OpenGL ou Direct3D.

2

#### Allegro

Tem funcionalidades para gráficos, sons, eventos e rotinas matemáticas sob uma licença gratuita para aplicações comerciais.

### 3 SFML (Simple and fast multimedia library)

É orientada a objetos. É simplificada para fácil uso, com módulos para sistema, janela, gráficos, áudio e rede. É uma multiplataforma com integração a várias linguagens.

4

### Raylib

É ideal para novatos e profissionais para prototipagem rápida e eficiente.

## Atividade 4

### Questão 1

Ao desenvolver aplicações multimídia e jogos, é essencial escolher uma biblioteca que ofereça acesso de baixo nível aos diversos componentes de hardware, como vídeo, áudio, teclado, mouse e joystick. Além disso, a capacidade de integrar gráficos 3D de alto desempenho é importante para criar experiências imersivas. Qual biblioteca em C é amplamente reconhecida por proporcionar essa gama de funcionalidades?

A

SFML (*simple and fast multimedia library*)

B

Allegro

C

SDL (*simple directmedia layer*)

D

Raylib

E

Irrlicht engine



A alternativa C está correta.

A SDL é uma biblioteca muito popular entre desenvolvedores de jogos em C, devido ao seu acesso direto e de baixo nível aos recursos do hardware. Ela suporta sistemas de vídeo e áudio e manipulação de entrada de dispositivos, sendo capaz de integrar gráficos 3D com o uso de OpenGL ou Direct3D. Ela é ideal para desenvolvedores que buscam controle detalhado sobre o hardware do sistema em suas aplicações de jogo.

## Programando com gráficos

Nesta prática, vamos explorar o uso da biblioteca `graphics.h` em C para criar uma animação simples: um círculo com bordas vermelhas se move horizontalmente de um lado para o outro na tela. A cada interação do círculo com as bordas da tela, há inversão na direção do movimento.

Confira neste vídeo conceitos-chave, como manipulação de eventos, controle de movimento e uso de cores, oferecendo uma introdução acessível e prática à programação gráfica por meio do uso da biblioteca `graphics.h` em C.

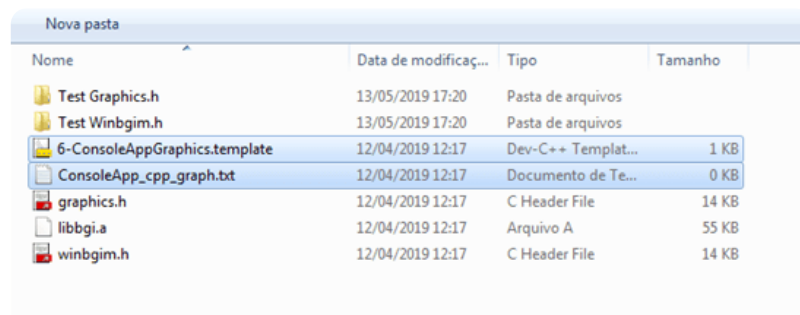


### Conteúdo interativo

Acesse a versão digital para assistir ao vídeo.

Nesta prática, vamos utilizar a biblioteca `graphics.h` para criar animações simples em C. Para isso, precisamos preparar o ambiente do DevC++ 5.11:

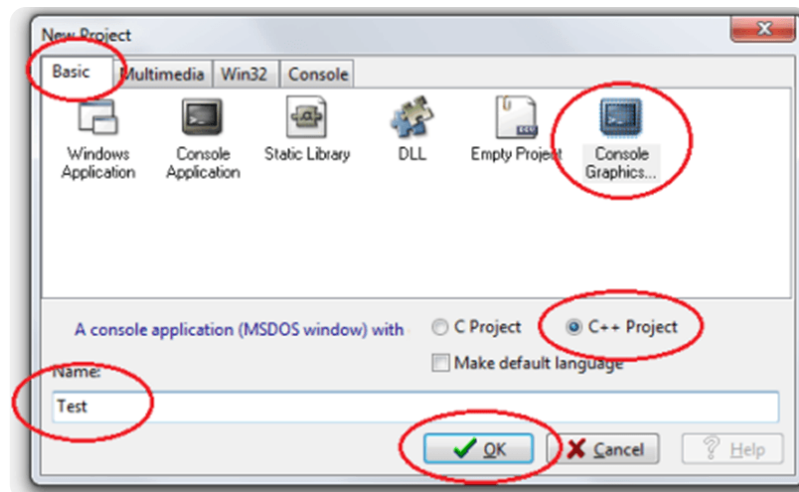
1. Baixe o driver `graphics.h`. Disponível [aqui](#).
2. Descompacte o arquivo Graphics in Dev C++.zip.
3. Copie os arquivos `6-ConsoleAppGraphics.template` e `ConsoleApp_cpp_graph.txt` (conforme demonstra a imagem a seguir) para a pasta `template` dentro daquela em que o Dev C++ está instalado. Em geral, isso ocorre nos arquivos de programa do Windows (`C:\Program Files (x86)\Dev-Cpp\Templates`).



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.txt	12/04/2019 12:17	Documento de Te...	0 KB
graphics.h	12/04/2019 12:17	C Header File	14 KB
libbgi.a	12/04/2019 12:17	Arquivo A	55 KB
winbgim.h	12/04/2019 12:17	C Header File	14 KB

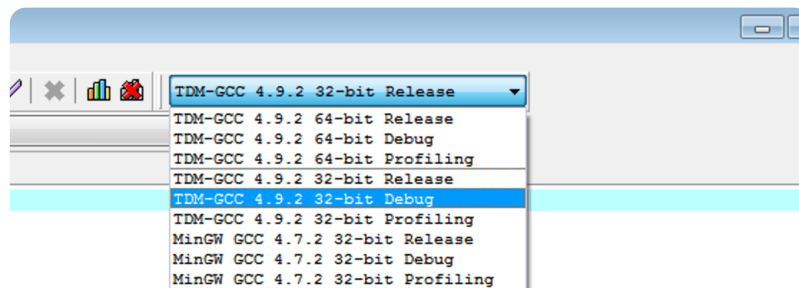
Tela do explorador de arquivos do Windows 10.

4. Copie os arquivos `graphics.h` e `winbgim.h` para a pasta `include` do compilador que será usado (32 bits) nos arquivos de programa do Windows (`C:\Program Files (x86)\Dev-Cpp\MinGW64\x86_64-w64-mingw32\include`).
5. Copie o arquivo `libbgi.a` para as pastas `lib` e `lib32` do compilador que será usado (32 bits) 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`).
6. Crie um projeto no Dev C++ com as opções `Console Graphics` e `C++ Project`, como mostrado na imagem a seguir (apesar de ser uma biblioteca de C, a opção `C project` não funciona).



Tela do Dev C++.

7. Escolha o compilador de 32 bits, como demonstrado na próxima imagem (TDM-GCC 4.9.2 32-bit Release).



Tela do Dev C++.

Agora, é possível escrever códigos que acessem esta biblioteca graphics.h no arquivo main.cpp do projeto. Confira!

```

c

#include
#include

int main() {
    int gd = DETECT, gm;
    int x = 100, y = 200; // Posição inicial do círculo
    int radius = 30;      // Raio do círculo
    int maxX;
    int step = 5;         // Passo do movimento do círculo

    initgraph(&gd, &gm, NULL); // Inicializa o sistema gráfico
    maxX = getmaxx(); // Pega a largura máxima da tela

    while (!kbhit()) { // Executa até que uma tecla seja pressionada
        cleardevice(); // Limpa a tela
        setcolor(RED); // Define a cor do círculo para vermelho
        circle(x, y, radius); // Desenha o círculo na posição atual

        // Atualiza a posição x do círculo
        x += step;

        // Inverte a direção quando atinge as bordas da janela
        if (x > maxX - radius || x < radius) {
            step = -step;
        }

        delay(50); // Espera um pouco antes de continuar
    }

    getch(); // Espera que o usuário pressione uma tecla
    closegraph(); // Fecha o sistema gráfico
    return 0;
}

```

## Faça você mesmo!

Você está trabalhando em um programa que utiliza a biblioteca `graphics.h` para criar a animação de um círculo vermelho que se move pela tela. Ele deve colidir com as bordas da janela e retornar para uma direção incerta. No entanto, o programa atual não funciona corretamente, pois não altera a direção de forma aleatória após a colisão. Veja como você pode corrigir isso.

```

c

#include
#include // Para rand() e srand()
#include // Para time()

int main() {
    int gd = DETECT, gm;
    int x = 300, y = 200; // Posição inicial do círculo
    int radius = 30;      // Raio do círculo
    int maxX, maxY;
    int dx = 4, dy = 4;   // Componentes de velocidade iniciais

    srand(time(NULL)); // Inicializa o gerador de números aleatórios

    initgraph(&gd, &gm, NULL); // Inicializa o sistema gráfico
    maxX = getmaxx(); // Pega a largura máxima da tela
    maxY = getmaxy(); // Pega a altura máxima da tela

    while (!kbhit()) { // Executa até que uma tecla seja pressionada
        cleardevice(); // Limpa a tela
        setcolor(RED); // Define a cor do círculo para vermelho
        circle(x, y, radius); // Desenha o círculo na posição atual

        // Atualiza a posição do círculo
        x += dx;
        y += dy;

        // Verifica se o círculo atinge as bordas e ajusta a direção
        if (x > maxX - radius || x < radius) {
            dx = -dx + (rand() % 4); // Inverte e ajusta aleatoriamente a direção
horizontal
        }
        if (y > maxY - radius || y < radius) {
            dy = -dy + (rand() % 4); // Inverte e ajusta aleatoriamente a direção vertical
        }

        delay(50); // Espera um pouco antes de continuar
    }

    getch(); // Espera que o usuário pressione uma tecla
    closegraph(); // Fecha o sistema gráfico
    return 0;
}

```

Considere que o trecho do programa responsável pelo problema é o seguinte:

```

// Verifica se o círculo atinge as bordas e ajusta a direção
if (x > maxX - radius || x < radius) {
    dx = -dx + (rand() % 4); // Inverte e ajusta aleatoriamente a direção horizontal
}
if (y > maxY - radius || y < radius) {
    dy = -dy + (rand() % 4); // Inverte e ajusta aleatoriamente a direção vertical
}

```

Qual das seguintes alterações corrigirá o problema no trecho destacado, garantindo que o círculo se choque na borda e retorne em uma direção aleatória?

A

Alterar `(rand() % 4)` para `(rand() % 8 - 4)` para permitir ajustes negativos e positivos na direção.

B

Remover `rand()` e simplesmente usar `dx = -dx;` e `dy = -dy;` para garantir a inversão correta.

C

Aumentar o intervalo de `rand() % 4` para `rand() % 10` para maior variação aleatória.

D

Substituir `dx = -dx + (rand() % 4);` por `dx = -dx * (rand() % 2 + 1);` para duplicar a velocidade ao inverter.

E

Usar `dx = dx + (rand() % 4);` sem inverter `dx`.



A alternativa A está correta.

A modificação proposta `(rand() % 8 - 4)` ajusta a formulação do código para incluir a possibilidade de ajustes tanto negativos como positivos na direção após a colisão. Isso inverte a direção e permite a variação aleatória, que pode ser positiva e negativa, mudando a trajetória do círculo de maneira mais dinâmica e imprevisível após cada colisão com as bordas da tela



## Conceitos básicos de OpenGL

API relevante e essencial para a renderização de gráficos 2D e 3D, a OpenGL é compatível com uma gama de dispositivos e sistemas operacionais. Compreender seus comandos, como desenho de polígonos e texturização — além do processo de pipeline gráfico — capacita qualquer desenvolvedor que deseja criar aplicações gráficas dinâmicas e visualmente impactantes.

Confira, neste vídeo, a funcionalidade e a aplicação de OpenGL, uma API extremamente portátil e rápida.



#### Conteúdo interativo

Acesse a versão digital para assistir ao vídeo.

OpenGL, abreviação para *open graphics library*, é uma API (ou seja, uma interface de programação de aplicativos) desenvolvida pela Silicon Graphics em 1992, mas mantida atualmente pelo Khronos Group. Essa API fornece um conjunto padrão de instruções para manipular gráficos em várias plataformas, o que permite aos desenvolvedores manter o mesmo código-base para gráficos em diferentes dispositivos como Macs, PCs e dispositivos móveis.

O OpenGL é também amplamente utilizado em jogos 3D, softwares CAD (como AutoCAD e Blender) e até a Apple utiliza essa API para suas bibliotecas gráficas no MacOS, como Core Animation e Core Image.

A principal vantagem do OpenGL é sua capacidade de operar em múltiplas plataformas, facilitando a transferência de softwares entre diferentes sistemas operacionais e ampliando seu uso para dispositivos móveis e aplicações web.

A API oferece comandos para o desenho de polígonos, atribuição de cores, texturização de polígonos, *zoom in* e *zoom out*, transformação e rotação de objetos. Ela também gerencia efeitos luminosos e atmosféricos, como nevoeiro, que podem ser aplicados a objetos individuais ou a cenas completas.



Laptop com imagem do programa AutoCAD na tela.

O processo de criação de imagens visíveis na tela é realizado por meio de:

- Pipeline gráfico, que envolve etapas como transformação e iluminação de vértices.
- Teste de profundidade.
- Aplicação de texturas.

As opções relatadas são uma série de passos, que permite personalização e otimização pelo desenvolvedor, melhorando o desempenho e a qualidade visual dos gráficos.

Por fim, OpenGL utiliza primitivas gráficas — como pontos, linhas e triângulos — para construir formas mais complexas. Vértices, que são pontos no espaço, definem essas primitivas e são armazenados em buffer objects na GPU para acesso rápido e eficiente.

Um componente importante na renderização moderna são os shaders, programas em GLSL que a GPU executa para controlar processos como transformação de vértices e definição de cores de pixels. A texturização, outra técnica importante, adiciona detalhes às superfícies dos objetos 3D, criando realismo por meio de cores, padrões ou relevo.

O OpenGL opera como uma máquina de estados, na qual os desenvolvedores podem configurar e modificar o comportamento da renderização para atender às necessidades específicas de cada aplicação. Com extensa compatibilidade e devido à possibilidade de ser estendida por várias extensões, o OpenGL continua a ser uma escolha popular entre os desenvolvedores gráficos ao redor do mundo, destacando-se por sua flexibilidade, eficiência e portabilidade.

## Atividade 1

### Questão 1

Em um curso de desenvolvimento de jogos, um aluno está aprendendo sobre as capacidades do OpenGL para manipular e renderizar gráficos 3D. Ao projetar um cenário de jogo que envolve mudanças de iluminação e efeitos atmosféricos (como nevoeiro), o aluno precisa escolher um comando do OpenGL que permita a implementação desses efeitos de forma eficaz.

Com base em seu conhecimento sobre OpenGL, qual das seguintes opções descreve um recurso que o aluno pode utilizar para adicionar efeitos atmosféricos no jogo?

A

Shaders, que são pequenos programas que permitem controlar a transformação de vértices e a cor dos pixels.

B

Buffer objects, que armazenam dados de vértices na GPU para acesso rápido.

C

Primitivas gráficas, como pontos e linhas, para construir as formas básicas dos objetos.

D

Funções de texturização, para aplicar detalhes visuais nas superfícies dos objetos.

E

Comandos para controle de efeitos luminosos e atmosféricos, como nevoeiro.



A alternativa E está correta.

O OpenGL fornece comandos que permitem que o desenvolvedor aplique alterações na iluminação e adicione efeitos que alteram a aparência ambiental, essenciais para criar a atmosfera desejada no jogo. As

outras opções, embora sejam características importantes do OpenGL, não se concentram especificamente na aplicação de efeitos atmosféricos.

## A biblioteca GLUT: interfaces de janelas

Vamos iniciar entendendo que a biblioteca GLUT é uma ferramenta para desenvolvedores que trabalham com a renderização de gráficos 3D usando OpenGL. Ao simplificar a interação com sistemas operacionais e dispositivos de entrada, a GLUT permite que os programadores se concentrem no design gráfico. Portanto, dominá-la é essencial para quem deseja se aprofundar na criação de aplicações gráficas sofisticadas e interativas.

Acompanhe, neste vídeo, a importância da biblioteca GLUT para desenvolvedores que utilizam OpenGL, detalhando como ela simplifica a criação e o gerenciamento de janelas e eventos de entrada. Veja também por que esse conhecimento é fundamental para quem deseja desenvolver aplicações gráficas interativas e eficientes.



### Conteúdo interativo

Acesse a versão digital para assistir ao vídeo.

A biblioteca GLUT, ou OpenGL utility toolkit, oferece solução eficaz para a integração de gráficos complexos com o gerenciamento de janelas e eventos de entrada. Embora o OpenGL seja primordial para criar gráficos, ele não gerencia a interação com o sistema operacional ou dispositivos de entrada, como mouse e teclado. A GLUT desempenha esse papel, agindo como uma camada intermediária que facilita as interações.

Ela permite que os desenvolvedores se concentrem na renderização gráfica, fornecendo uma interface simplificada para criar e gerenciar janelas e tratar eventos de entrada. Isso permite a configuração de um contexto gráfico para a renderização das primitivas do OpenGL, isolando o programador das especificidades do sistema operacional utilizado.

Em termos práticos, utilizar o GLUT é como ter um assistente técnico para o desenvolvimento de software gráfico, simplificando a abertura de múltiplas janelas e o gerenciamento de eventos, como redimensionamentos e interações do usuário.

A GLUT é reconhecida por sua simplicidade e robustez, que exigem poucas rotinas para exibir uma cena gráfica e proporcionam estabilidade devido à sua ampla utilização. Além disso, ela suporta callbacks, funções acionadas por eventos específicos, que são essenciais em aplicações como jogos ou programas interativos, em que a resposta rápida aos comandos do usuário é essencial.

O design da API GLUT facilita a programação ao evitar complicações comuns, como o uso extensivo de ponteiros ou identificadores de sistema, e trata as coordenadas de janela de forma intuitiva, simplificando o posicionamento dos elementos gráficos.

Para desenvolvedores iniciantes ou experientes que buscam eficiência, GLUT oferece uma base sólida e acessível para a criação de interfaces gráficas robustas e interativas no ambiente OpenGL.

## Atividade 2

### Questão 1

João é um desenvolvedor de jogos digitais e decidiu usar o OpenGL para criar gráficos 3D em seu novo projeto. Ele optou pelo uso da biblioteca GLUT. Analise as alternativas a seguir:

Asserção: o uso da biblioteca GLUT por João é adequado para simplificar a criação e o gerenciamento de janelas e eventos de entrada em seu projeto de jogo.

Razão: a biblioteca GLUT oferece uma camada de abstração que permite aos desenvolvedores se concentrarem na renderização gráfica sem se preocuparem com detalhes específicos do sistema operacional ou gerenciamento de janelas.

Selecione, a seguir, a alternativa correta.

A

Ambas são verdadeiras, e a razão é a explicação correta da asserção.

B

Ambas são verdadeiras, mas a razão não é a explicação correta da asserção.

C

A asserção é verdadeira, mas a razão é falsa.

D

A asserção é falsa, mas a razão é verdadeira.

E

Ambas são falsas.



A alternativa A está correta.

A asserção corretamente identifica que a biblioteca GLUT é uma escolha adequada para João, pois ela simplifica a interação com o sistema de janelas e o tratamento de eventos de entrada, o que é importante no desenvolvimento de jogos. A razão justifica adequadamente essa escolha, explicando que a GLUT fornece uma interface que abstrai complexidades específicas de sistemas operacionais, permitindo que o desenvolvedor se concentre mais na parte gráfica do projeto.

## O ambiente Dev C++

Quem deseja se aprofundar na programação gráfica usando um IDE gratuito e leve deve explorar a configuração do Dev-C++ para utilizar a biblioteca OpenGL e GLUT. Dev-C++ oferece um ambiente desenvolvido e intuitivo para configurar essas bibliotecas, permitindo que desenvolvedores criem gráficos 2D e 3D com facilidade.

Veja, neste vídeo, o processo de instalação e configuração de diretórios e como preparar o sistema para a execução de programas gráficos.



### Conteúdo interativo





Acesse a versão digital para assistir ao vídeo.

O Dev-C++ na versão 5.11 é um IDE gratuito e de código aberto para C/C++, ideal para estudantes e iniciantes devido à sua simplicidade e eficácia. Ele inclui compilador, depurador, editor de código e suporta bibliotecas como OpenGL para gráficos 3D. Para usar GLUT e OpenGL, são necessárias configurações iniciais no Dev-C++, facilitando o desenvolvimento de aplicações gráficas.

Acompanhe o passo a passo a seguir!

#### 1. Instalação da biblioteca GLUT:

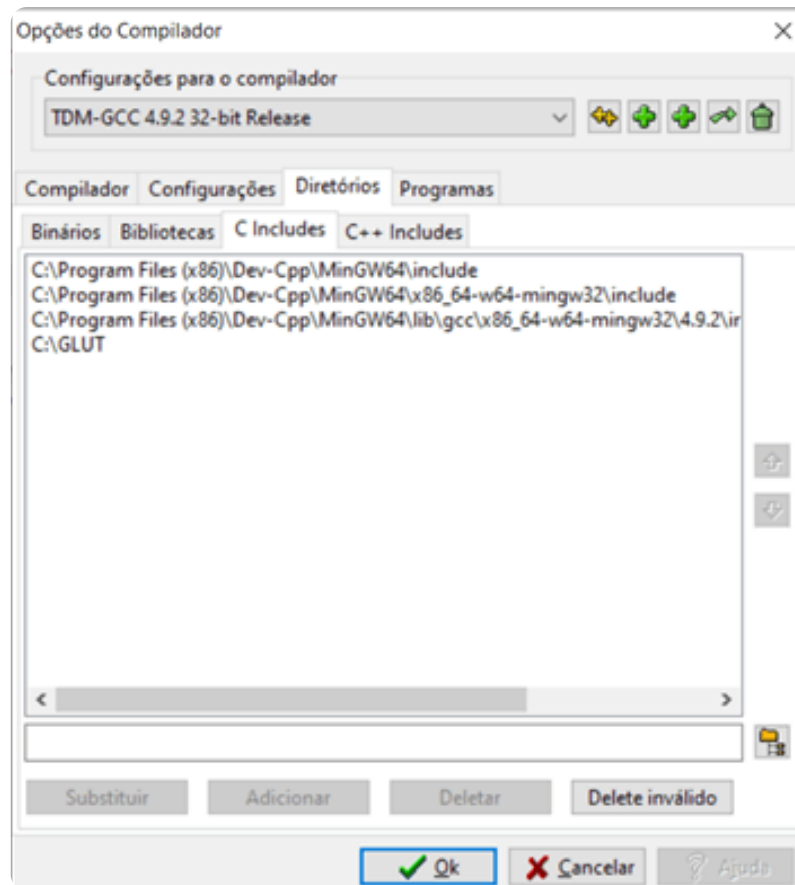
- Baixe a biblioteca GLUT, disponível [aqui](#).
- Extraia os arquivos baixados para a pasta C:\GLUT (como demonstrado na imagem a seguir). Isso inclui arquivos essenciais como glut.h (header file).

Nome	Tipo	Tamanho
 glut.h	C Header File	21 KB
 glut32.def	Arquivo DEF	3 KB
 glut32.dll	Extensão de aplica...	204 KB
 libglut.a	Arquivo A	93 KB

Tela do explorador de arquivos do Windows 10.

#### 2. Configuração do Dev-C++:

- Abra o Dev-C++ e navegue até Ferramentas → Opções do compilador.
- Adicione, Na aba Diretórios, a pasta C:\GLUT às pastas de busca do compilador.
- Clique no ícone com uma seta (como demonstrado na tela a seguir) e selecione C:\GLUT.



Tela do Dev C++.

- Adicione a pasta tanto nas abas Bibliotecas como em C Includes para que o compilador saiba onde buscar os arquivos de biblioteca e os headers durante a compilação.

### 3. Adição do arquivo DLL:

Copie o arquivo glut32.dll para a pasta C:\Windows\system32 ou C:\Windows\SysWOW64 dependendo da versão do seu sistema operacional (32 bits ou 64 bits, respectivamente). Você também pode colocar o arquivo diretamente no diretório do projeto para evitar problemas de permissão.

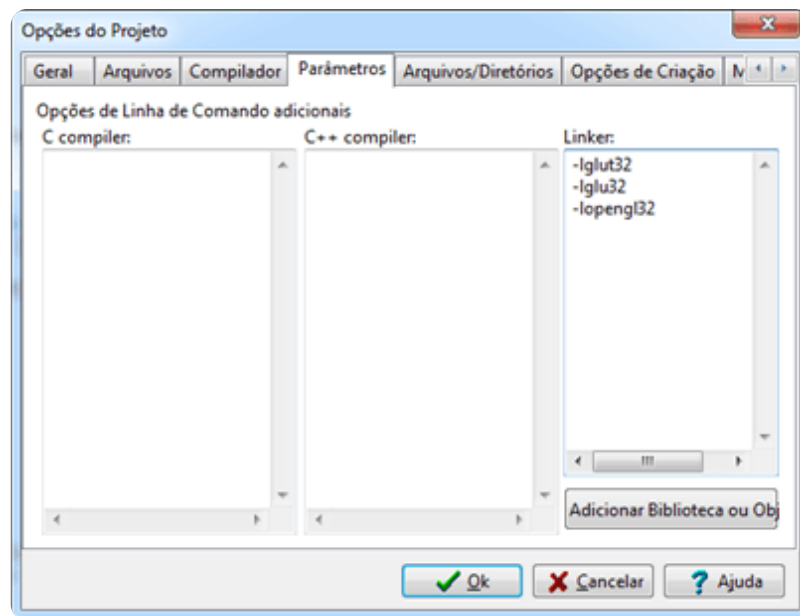
### 4. Criação e configuração do projeto:

- Crie, no Dev-C++, um novo projeto selecionando Arquivo → Novo → Projeto e escolha Console application e Projeto C.
- Acesse Projeto → Opções do projeto, e na aba Parâmetros, na coluna Linkers, adicione os seguintes comandos:

-lglut32

-lglu32

-lopengl32



Tela do Dev C++.

Os comandos vistos garantem que o linker saiba quais bibliotecas adicionar ao compilar seu projeto.

##### 5. Teste sua configuração:

Experimente o ambiente com um código básico de OpenGL para verificar se tudo está funcionando corretamente. No arquivo principal do projeto (main.c), acrescente o código a seguir. Deverá aparecer uma janela (com fundo preto e um quadrado branco) e o título Ola Mundo:



Acompanhe o código:

```
c

#include
void display(void)
{
    glClear(GL_COLOR_BUFFER_BIT);
    glBegin(GL_POLYGON);
    glVertex2f(-0.3, -0.3);
    glVertex2f(-0.3, 0.3);
    glVertex2f(0.3, 0.3);
    glVertex2f(0.3, -0.3);
    glEnd();
    glFlush();
}
int main(void) {
    glutInitDisplayMode(GLUT_SINGLE | GLUT_RGB);
    glutCreateWindow("Ola Mundo");
    glutDisplayFunc(display);
    glutMainLoop();
}
```

## Atividade 3

### Questão 1

Um programador está trabalhando em uma aplicação de design assistido por computador (CAD) que requer renderização gráfica avançada. Ele escolheu usar a biblioteca OpenGL pelo ambiente de desenvolvimento Dev-C++. Depois de configurar corretamente o Dev-C++ para incluir a biblioteca GLUT, qual é o próximo passo para garantir que os programas OpenGL sejam executados sem problemas?

A

Compilar o código fonte usando o modo de compatibilidade do Windows.

B

Copiar o arquivo glut32.dll para uma das pastas do sistema Windows adequadas.

C

Alterar as configurações de energia do computador para alto desempenho.

D

Reinstalar o Dev-C++ para garantir que todas as configurações sejam padrão.

E

Converter todos os projetos antigos para a versão mais recente do Dev-C++.





A alternativa B está correta.

O passo essencial após configurar a biblioteca GLUT no ambiente Dev-C++ é garantir que o arquivo DLL, especificamente o glut32.dll, esteja na pasta adequada do sistema Windows. Isso é necessário porque o sistema operacional precisa acessar essa biblioteca dinâmica durante a execução de aplicativos que usam OpenGL. Colocar o glut32.dll na pasta do sistema ou no diretório do projeto evita erros de arquivo não encontrado ao iniciar o aplicativo.

## Mecanismo de máquina de estados do OpenGL

Vamos iniciar entendendo de que modo OpenGL opera: como uma poderosa máquina de estados, na qual diversas variáveis — conhecidas como o contexto do OpenGL — configuram seu funcionamento. Compreender esse mecanismo permite manipular eficazmente a API, uma vez que ajustes nos estados influenciam diretamente a renderização gráfica.

Acompanhe, neste vídeo, a estrutura e o funcionamento do OpenGL como uma máquina de estados, explicando como as variáveis de contexto influenciam a renderização.



### Conteúdo interativo

Acesse a versão digital para assistir ao vídeo.

Ao utilizar OpenGL, o estado é modificado ajustando configurações, manipulando buffers e executando operações baseadas no contexto vigente. Por exemplo, ao mudarmos o modo de desenho de triângulos para linhas, alteramos variáveis de contexto, afetando os comandos de desenho subsequentes que passarão a produzir linhas.

Programar com OpenGL envolve tanto alterar estados como utilizar o estado atual para realizar operações. Entender essa dinâmica é essencial, pois um gerenciamento eficiente dos estados serve para otimizar o desempenho de aplicações gráficas, evitando mudanças frequentes e desnecessárias que podem reduzir a eficiência da renderização.

## Contexto de máquina de estados no programa exemplo

Vejamos o código exemplo que cria uma janela e desenha um polígono dentro dela. Vamos examinar cada parte do código e como ele se relaciona com o mecanismo de máquina de estados da OpenGL.

### 1. Inclusão da biblioteca GLUT

```
c
#include
```

A linha insere a biblioteca GLUT, que fornece funções utilitárias para criar janelas e gerenciar interações do usuário, como teclado e mouse, em aplicações OpenGL.

## 2. Função display

```
c
void display(void)
{
    glClear(GL_COLOR_BUFFER_BIT);
    glBegin(GL_POLYGON);
    glVertex2f(-0.3, -0.3);
    glVertex2f(-0.3, 0.3);
    glVertex2f(0.3, 0.3);
    glVertex2f(0.3, -0.3);
    glEnd();
    glFlush();
}
```

Define o que será desenhado na janela:

- `glClear(GL_COLOR_BUFFER_BIT)`: limpa o buffer de cor da janela para preparar para o novo desenho. Configura um estado do OpenGL para limpeza de tela.
- `glBegin(GL_POLYGON)` até `glEnd()`: delimitam a definição de um polígono. Entre `glBegin` e `glEnd`, chamadas de `glVertex2f` especificam os vértices do polígono. Modifica o estado de renderização para definir a geometria que será desenhada.
- `glFlush()`: garante que todos os comandos de OpenGL anteriores sejam executados o mais rápido possível. Manipula o estado de buffer.

## 3. Função main

```
c
int main(void)
{
    glutInitDisplayMode(GLUT_SINGLE | GLUT_RGB);
    glutCreateWindow("Ola Mundo");
    glutDisplayFunc(display);
    glutMainLoop();
}
```

Configura a janela e o loop de exibição. Confira mais detalhes:

- `glutInitDisplayMode(GLUT_SINGLE | GLUT_RGB)`: configura o modo de exibição da janela. `GLUT_SINGLE` indica o uso de um único buffer de exibição, e `GLUT_RGB` define que a janela utilizará um modelo de cor RGB. Isso configura os estados da janela de exibição.
- `glutCreateWindow("Ola Mundo")`: cria uma janela com o título "Ola Mundo".
- `glutDisplayFunc(display)`: registra a função `display` como a de callback que será chamada para redesenhar a janela. Isso não altera diretamente o estado do OpenGL, mas define como o estado deve ser gerenciado quando a janela precisa ser atualizada.
- `glutMainLoop()`: inicia o loop de processamento de eventos da GLUT, que gerencia a exibição e os outros eventos de entrada, mantendo a aplicação ativa.

O programa utiliza o mecanismo de máquina de estados do OpenGL ao definir e alterar estados para controlar a renderização. Cada chamada de função — como `glClear`, `glBegin`, `glVertex2f` e `glEnd` — modifica o estado do OpenGL para realizar a tarefa desejada de renderização. O estado é mantido até ser explicitamente alterado por outra chamada de função, demonstrando o princípio da máquina de estados que a OpenGL adota.

## Atividade 4

### Questão 1

Em um projeto de desenvolvimento de jogos usando OpenGL, um programador está tentando otimizar o processo de renderização. Ele precisa decidir qual estratégia adotar para minimizar a frequência de alterações de estado, conhecidas por potencialmente reduzirem a eficiência do sistema. Qual das seguintes abordagens é a mais apropriada para otimizar o desempenho da renderização no OpenGL?

A

Alterar o estado de renderização após cada chamada de desenho para garantir a precisão.

B

Manter o número mínimo de mudanças de estado, agrupando operações similares.

C

Utilizar múltiplos contextos de OpenGL para cada tipo de objeto a ser renderizado.

D

Redefinir frequentemente o estado de renderização para evitar conflitos de buffer.

E

Ignorar o estado da máquina até que ocorram erros de renderização, para, só então, ajustar.



A alternativa B está correta.

A eficiência na renderização no OpenGL pode ser severamente afetada por mudanças frequentes e desnecessárias no estado. Portanto, a melhor abordagem é minimizar as mudanças agrupando operações que compartilham configurações de estado similares. Isso reduz o *overhead* associado à configuração de estados e melhora o desempenho geral do sistema.

## Utilizando OpenGL na prática

Nesta prática de programação, você aprenderá a configurar um ambiente de desenvolvimento para OpenGL usando a biblioteca GLUT e criará um aplicativo simples que renderiza um cubo 3D rotativo. Você configurará a janela, definirá parâmetros de iluminação e câmera e implementará a animação do cubo por meio de rotações contínuas.

Você também manipulará a matriz de transformações para entender como objetos 3D são dispostos e orientados no espaço. Essa atividade prática visa fornecer compreensão básica, mas completa, dos princípios de renderização 3D usando OpenGL, o que é essencial para qualquer desenvolvedor gráfico que deseje explorar mais a fundo a criação de gráficos computacionais avançados.

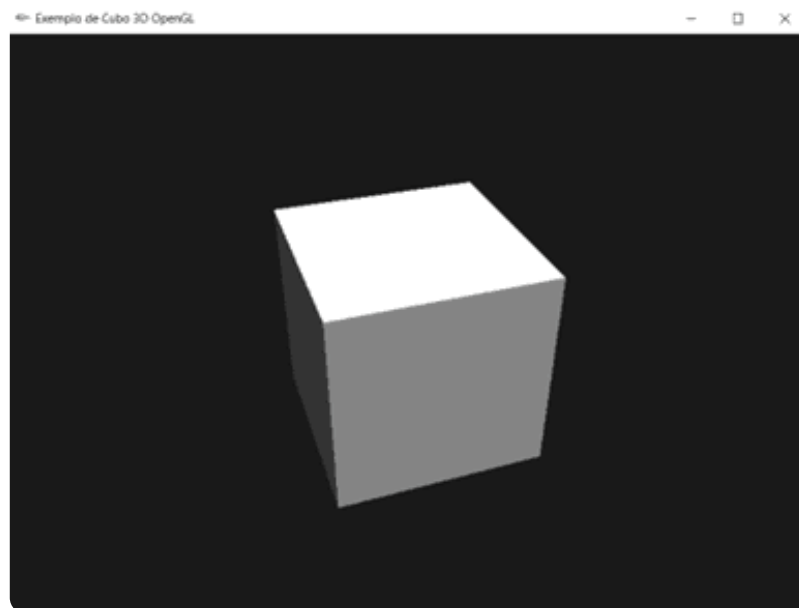
Acompanhe, neste vídeo, a configuração e o uso do OpenGL com GLUT para criar um cubo 3D rotativo, demonstrando a importância do controle de estados e matrizes de transformação. A prática incluirá ajustes de velocidade de rotação, iluminação e câmera, fornecendo uma base para futuros projetos de renderização 3D. Vale a pena conferir!



#### Conteúdo interativo

Acesse a versão digital para assistir ao vídeo.

Para essa prática em OpenGL com GLUT, assumiremos que você já tem o ambiente Dev-C++ configurado para as APIs em seu computador. Usando essa ferramenta, você aprenderá a criar um aplicativo que renderiza um cubo 3D rotativo. O resultado pode ser visto na imagem a seguir.



Tela do programa OpenGL com GLUT, em ambiente Dev-C++.

Veja o código completo:

```

c

#include

// Função chamada para desenhar
void display() {
    glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);
    glLoadIdentity();

    // Posiciona a câmera
    gluLookAt(3.0, 3.0, 3.0, 0.0, 0.0, 0.0, 0.0, 1.0, 0.0);

    // Rotação simples ao redor do eixo Y
    static float angle = 0.0;
    glRotatef(angle, 0.0, 1.0, 0.0);

    // Desenha o cubo
    glutSolidCube(1.5);

    // Atualiza o ângulo
    angle += 0.2;
    if (angle > 360) angle = 0;

    glutSwapBuffers();
}

// Inicializa parâmetros de rendering
void initRendering() {
    glEnable(GL_DEPTH_TEST);
    glEnable(GL_LIGHTING);
    glEnable(GL_LIGHT0);
    glEnable(GL_NORMALIZE);
    glEnable(GL_COLOR_MATERIAL);

    // Define a cor do fundo
    glClearColor(0.1, 0.1, 0.1, 1.0);

    // Configuração da iluminação
    GLfloat lightPos[] = {1.0, 1.0, 1.0, 0.0};
    glLightfv(GL_LIGHT0, GL_POSITION, lightPos);
}

// Chamado quando a janela é redimensionada
void handleResize(int w, int h) {
    glViewport(0, 0, w, h);
    glMatrixMode(GL_PROJECTION);
    glLoadIdentity();
    gluPerspective(45.0, (double)w / (double)h, 1.0, 200.0);
    glMatrixMode(GL_MODELVIEW);
    glLoadIdentity();
}

int main(int argc, char** argv) {
    // Inicializa GLUT e processa os argumentos do usuário
    glutInit(&argc, argv);
    glutInitDisplayMode(GLUT_DOUBLE | GLUT_RGB | GLUT_DEPTH);

    // Configura o tamanho e a posição inicial da janela
    glutInitWindowSize(800, 600);
    glutInitWindowPosition(100, 100);

    // Cria a janela
    glutCreateWindow("Exemplo de Cubo 3D OpenGL");

    // Inicializa alguns parâmetros de rendering
    initRendering();

    // Configura as funções de callback
    glutDisplayFunc(display);
}

```

Vamos analisar o código em detalhes. Vamos lá!

### Função display

- `glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT)`: limpa os buffers de cor e profundidade para preparar para um novo quadro, garantindo que não haja resíduos visuais de desenhos anteriores.
- `glLoadIdentity()`: restaura a matriz de transformação para o estado padrão (identidade), garantindo que as transformações não se acumulem entre os quadros.
- `gluLookAt(3.0, 3.0, 3.0, 0.0, 0.0, 0.0, 0.0, 1.0, 0.0)`: define a posição da câmera. Os três primeiros valores são a posição da câmera no espaço, os três seguintes são as coordenadas para onde ela está olhando, e os últimos três definem a orientação vertical da câmera.
- `glRotatef(angle, 0.0, 1.0, 0.0)`: aplica uma rotação em torno do eixo Y (vertical) ao cubo, no qual angle é o ângulo de rotação.
- `glutSolidCube(1.5)`: desenha um cubo sólido com comprimento de aresta 1.5 unidades.
- `angle += 0.2`: atualiza o ângulo para a próxima renderização, incrementando-o pela velocidade definida.
- `if (angle > 360) angle -= 360`: garante que o valor do ângulo não cresça indefinidamente.
- `glutSwapBuffers()`: troca os buffers de fundo para os de frente, exibindo o quadro renderizado e iniciando o desenho do próximo quadro no buffer oculto.

### Função initRendering

- `glEnable(GL_DEPTH_TEST)`: habilita o teste de profundidade, que ajuda a garantir que superfícies sejam corretamente ocultadas por outras que estão à frente delas.
- `glEnable(GL_LIGHTING)` e `glEnable(GL_LIGHT0)`: habilitam a iluminação e a primeira fonte de luz, respectivamente, para adicionar efeitos de luz e sombra.
- `glEnable(GL_NORMALIZE)` e `glEnable(GL_COLOR_MATERIAL)`: as normais são vetores perpendiculares à superfície dos objetos em um cenário 3D. Eles são essenciais para o cálculo da iluminação, pois determinam como a luz interage com a superfície. Em OpenGL, quando um objeto é transformado (escalado, rotacionado ou transladado), suas normais também sofrem transformações. Para assegurar que elas sejam corretamente ajustadas e continuem a representar vetores unitários (de comprimento 1), utiliza-se o comando **`glEnable(GL_NORMALIZE)`**. A função `glEnable(GL_COLOR_MATERIAL)` permite que as cores dos materiais dos objetos reajam corretamente à iluminação.
- `glClearColor(0.1, 0.1, 0.1, 1.0)`: define a cor de fundo.
- `glLightfv(GL_LIGHT0, GL_POSITION, lightPos)`: estabelece a posição da fonte de luz.

### Função handleResize

- `glViewport(0, 0, w, h)`: ajusta a área de desenho para o tamanho atual da janela.
- `glMatrixMode(GL_PROJECTION)` e `glMatrixMode(GL_MODELVIEW)`: alternam entre modificar a matriz de projeção e a matriz de modelo/vista, respectivamente.
- `gluPerspective(45.0, (double)w / (double)h, 1.0, 200.0)`: define a perspectiva da câmera com um campo de visão de 45 graus e um aspecto baseado no tamanho da janela.

### Função main

- glutInit e funções relacionadas configuram a janela e o modo de exibição.
- glutDisplayFunc(display) e glutIdleFunc(display): definem a função display como a função de callback para desenhar e atualizar a janela.
- glutMainLoop(): inicia o loop de eventos da GLUT.

## Faça você mesmo!

### Questão 1

No contexto do projeto usando OpenGL para renderizar um cubo 3D rotativo, você ficou responsável por ajustar a velocidade de rotação do cubo para torná-la mais lenta, a fim de facilitar a visualização detalhada de suas faces. Qual das seguintes alternativas representa a melhor abordagem para você conseguir isso?

A

Aumentar o valor de incremento do ângulo de rotação a cada atualização.

B

Diminuir o valor de incremento do ângulo de rotação a cada atualização.

C

Reduzir a taxa de quadros por segundo (FPS) na aplicação.

D

Adicionar uma pausa entre cada incremento do ângulo de rotação.

E

Reduzir a resolução da janela de visualização.



A alternativa B está correta.

A velocidade de rotação de um objeto em OpenGL é diretamente influenciada pelo incremento aplicado ao ângulo de rotação em cada ciclo de atualização do quadro. Diminuindo esse valor, a alteração do ângulo por ciclo será menor, resultando em uma rotação mais lenta.

## O conceito de eventos

Em programação, especialmente em linguagem C, a captura de eventos é um fundamento crítico para desenvolvedores que trabalham com interfaces gráficas ou dispositivos interativos. Esse conceito permite que programas respondam dinamicamente às ações do usuário, como cliques de mouse ou pressionamentos de teclas, tornando-as essenciais para a criação de software interativo e funcional.

Acompanhe, neste vídeo, as práticas de implementação de eventos em programação com exemplos reais, como detecção de cliques do mouse em aplicações Windows. Tente compreender a importância desse conhecimento para o desenvolvimento de um software eficaz.



### Conteúdo interativo

Acesse a versão digital para assistir ao vídeo.

No contexto da linguagem C, que é bastante utilizada para programação de sistemas e aplicativos com necessidade de controle fino sobre os recursos do hardware, entender como capturar e responder a esses eventos pode ser um diferencial para a criação de programas interativos.

Imagine que você está desenvolvendo um software para uma loja. O caixa precisa de uma interface em que possa registrar produtos simplesmente os escaneando. Cada vez que um produto é lido, um evento (nesse caso, o escaneamento) é gerado. O seu programa em C deve estar pronto para capturar esse evento e processar as informações, por exemplo, adicionando o produto a uma lista de compras e calculando o total.

Para capturar e processar eventos de hardware no sistema operacional Windows, uma biblioteca amplamente utilizada é a Windows API (WinAPI). Ela fornece uma série de funções para manipulação de eventos de entrada, como teclado e mouse, e é adequada para capturar eventos de dispositivos, como scanners de código de barras.



Mulher escaneando produtos encaixotados.

## Utilização da Windows API (WinAPI)

É importante definir que WinAPI é um conjunto de interfaces que permite a quem desenvolve interagir com o sistema operacional Windows. Veja um exemplo básico de como usar a WinAPI para capturar eventos de teclado, podendo ser adaptada para capturar eventos de scanners que emulam entradas de teclado:



```

c

#include
#include

// Callback para tratar eventos de teclado
LRESULT CALLBACK KeyboardProc(int nCode, WPARAM wParam, LPARAM lParam) {
    if (nCode == HC_ACTION) {
        if (wParam == WM_KEYDOWN) {
            KBDLLHOOKSTRUCT *p = (KBDLLHOOKSTRUCT *)lParam;
            printf("Key pressed: %d\n", p->vkCode);
            // Processar evento de escaneamento aqui
        }
    }
    return CallNextHookEx(NULL, nCode, wParam, lParam);
}

int main() {
    // Configurar um hook de teclado global
    HHOOK keyboardHook = SetWindowsHookEx(WH_KEYBOARD_LL, KeyboardProc, NULL, 0);
    if (keyboardHook == NULL) {
        printf("Failed to install hook!\n");
        return 1;
    }

    // Loop de mensagens para manter o hook ativo
    MSG msg;
    while (GetMessage(&msg, NULL, 0, 0)) {
        TranslateMessage(&msg);
        DispatchMessage(&msg);
    }

    // Remover o hook quando terminar
    UnhookWindowsHookEx(keyboardHook);
    return 0;
}

```

No exemplo, o programa instala um hook de teclado global usando a função `SetWindowsHookEx` com o tipo de hook `WH_KEYBOARD_LL`, que é de baixo nível para capturar eventos de teclado. A função `KeyboardProc` é utilizada toda vez que uma tecla é pressionada. Dentro dessa função, podemos processar o evento conforme necessário, como adicionar o produto escaneado a uma lista de compras.

A Windows API oferece uma abordagem desenvolvida e flexível para capturar e processar eventos de hardware em ambientes Windows. Isso permite que desenvolvedores criem aplicações interativas, responsivas e capazes de se integrar facilmente a dispositivos de entrada, como scanners de código de barras.

## Atividade 1

### Questão 1

Em uma aplicação de controle de segurança desenvolvida em C para uma empresa, é essencial que o software responda a eventos específicos, como a detecção de um cartão RFID apresentado a um leitor. O programa deve capturar e processar esse evento para permitir ou negar o acesso. Considerando a importância de responder a esses eventos, qual das seguintes bibliotecas da API do Windows seria a melhor para ajudar na captura e no tratamento desses eventos?

A

Windows Forms

B

WinAPI

C

MFC (*Microsoft foundation classes*)

D

DirectX

E

OpenGL



A alternativa B está correta.

A WinAPI (Windows application programming interface) é fundamental para desenvolver aplicações que precisam interagir de forma direta e eficiente com o hardware e o sistema operacional Windows. Ela permite manipulação detalhada de eventos de hardware, como a leitura de um cartão RFID.

## A Interrupção em programação

Vamos começar entendendo que, para quem deseja interagir com hardware e criar softwares responsivos, a interrupção é um conceito importante em programação, especialmente em C. Compreender a diferença em relação aos eventos é útil para aprimorar o aprendizado em sistemas embarcados e operacionais e aplicar adequadamente suas funcionalidades em situações práticas, garantindo programas eficientes e adaptados às necessidades do usuário.

Neste vídeo, exploraremos a importância das interrupções em programação, diferenciando-as dos eventos, com ênfase em suas aplicações práticas, como na programação de sistemas embarcados e dispositivos interativos.



### Conteúdo interativo

Acesse a versão digital para assistir ao vídeo.

Uma interrupção é como um alarme urgente que manda o processador parar temporariamente o que está fazendo e prestar atenção em algo mais importante. Ela é usada principalmente para lidar com eventos de hardware. Por exemplo, se você está digitando em um teclado, cada pressionamento de tecla gera uma interrupção que diz ao CPU para ler a tecla pressionada e fazer algo com essa informação.

Em termos técnicos, uma interrupção é um sinal enviado ao processador, seja de software ou hardware, que interrompe o fluxo normal de execução do programa. O processador responde imediatamente à interrupção,

salvando seu estado atual e executando uma função especial chamada **rotina de serviço de interrupção (ISR)**, que trata especificamente do evento que causou a parada.



### Exemplo

Em um sistema embarcado, como um microcontrolador em um drone, uma interrupção pode ser usada para processar o sinal de um sensor de altitude que indica que o aparelho está muito próximo ao chão, exigindo correção imediata de altitude.

## Diferença entre evento e interrupção

São conceitos fundamentais em programação, especialmente em linguagem C, que é amplamente usada para programar sistemas que interagem diretamente com o hardware, como sistemas embarcados e operacionais. Apesar dos conceitos de evento e interrupção serem mecanismos para gerenciar como um programa responde a ações externas, eles têm diferenças importantes na maneira como são implementados e postos em uso.

A principal diferença entre interrupções e eventos está na urgência e no nível de atenção que o processador precisa dedicar. Interrupções são imediatas e prioritárias, pois podem ocorrer a qualquer momento, e precisam ser tratadas o mais rápido possível. Já os eventos podem ser tratados de forma mais flexível, sendo processados dentro do fluxo normal do programa.

No dia a dia profissional, a pessoa desenvolvedora de sistemas embarcados pode usar interrupções para garantir que o sistema responda prontamente a condições críticas, como paradas de emergência em máquinas industriais ou sinais de sensores em um dispositivo médico.

Eventos, por sua vez, são úteis para criar interfaces de usuário interativas em softwares ou para gerenciar tarefas que não exigem resposta imediata, como atualizar um display ou processar dados de entrada em um formulário.

Manipular interrupções de hardware diretamente em sistemas operacionais modernos, como Windows ou Linux, requer privilégios especiais e normalmente é gerenciado pelo próprio sistema operacional. Usando C, somente programas para Windows 16-bits e DOS permitem o acesso aos registros da arquitetura para manipulação de interrupções.

## Atividade 2

### Questão 1

No contexto de desenvolvimento de sistemas embarcados, considere um drone que utiliza interrupções para responder a sinais críticos, como a proximidade ao solo. Qual das seguintes afirmações descreve corretamente a função de uma interrupção no gerenciamento de hardware?

A

Interrupções são ativadas apenas por eventos de software.

B

Interrupções e eventos têm a mesma urgência e forma de tratamento pelo processador.

C

Interrupções processam eventos não urgentes, como atualizações de display.

D

Uma interrupção faz o processador parar para responder urgentemente a condições de hardware, como sensores de proximidade.

E

Interrupções são desnecessárias em sistemas embarcados modernos, já que o hardware os gerencia automaticamente.



A alternativa D está correta.

A função de uma interrupção é forçar o processador a pausar suas atividades atuais e responder a situações de hardware urgentes. As outras alternativas são incorretas por distorcerem a natureza e a urgência das interrupções em comparação aos eventos ou por sugerirem redução gradual incorreta das interrupções em tecnologias modernas.

## Captura de eventos de teclado e mouse

Para desenvolver aplicativos interativos no Windows, é essencial capturar eventos de teclado e mouse em C. Dominar essa habilidade prática, por meio do entendimento e da aplicação do código usando windows.h, é importante para quem deseja criar softwares responsivos, sendo um fundamento básico no aprendizado de programação.

Confira, neste vídeo, como configurar o ambiente e o procedimento de janela para responder às interações do usuário, destacando a importância prática dessas habilidades.



Conteúdo interativo

Acesse a versão digital para assistir ao vídeo.

Capturar eventos de teclado e mouse na programação em C é uma tarefa comum para muitos desenvolvedores, especialmente aqueles envolvidos na criação de aplicativos interativos para Windows. Quando lidamos com esses eventos, essencialmente damos aos nossos programas a capacidade de responder a ações do usuário, como cliques e teclas pressionadas.

Utilizaremos a biblioteca windows.h para acessar essas funcionalidades, o que é bastante prático se você estiver usando um ambiente de desenvolvimento como o Dev-C++ no Windows.

## Entendendo o básico sobre a captura de eventos de teclado e mouse

Quando falamos de captura de eventos de teclado e mouse, estamos nos referindo à interceptação e ao manejo dessas ações dentro do nosso código. Isso é feito por meio de uma função chamada **procedimento de janela**, que recebe mensagens do sistema sobre vários eventos, como pressionamento de teclas ou movimentos do mouse.

## Exemplo prático

Imagine que você está desenvolvendo um jogo simples ou uma aplicação que precisa de inputs do usuário. Veremos neste material um exemplo de como você poderia configurar o procedimento de janela para lidar com esses eventos.

O código apresentado é um exemplo simples de uma aplicação Windows, que detecta cliques de mouse, especificamente para os botões esquerdo e direito. Ele foi escrito em C utilizando a API do Windows.

Acompanhe os passos para configurar o Dev-C++:

- Abra o Dev-C++, selecione Arquivo > Novo > Projeto... e escolha Projeto do Windows para garantir que as bibliotecas corretas sejam vinculadas automaticamente.
- Adicione o código a seguir no arquivo principal:

c

```
#include
/* inclui o arquivo de cabeçalho windows.h, que contém declarações
para todas as funções da API do Windows, tipos de dados e constantes
necessárias para criar aplicações Windows.*/

// Protótipo da função de tratamento de mensagens da janela
LRESULT CALLBACK WindowProcedure(HWND, UINT, WPARAM, LPARAM);
/*Declara um protótipo para a função de procedimento da janela.
Esta função é chamada pela API do Windows para tratar mensagens direcionadas para a
janela,
como cliques do mouse, movimentos, fechamento da janela, etc.*/

char szClassName[] = "WindowsApp";

/*WinMain é o ponto de entrada para uma aplicação Windows,
equivalente à função main em programas C padrão.
Esta função é chamada pelo sistema operacional quando a aplicação é iniciada.*/
int WINAPI WinMain(HINSTANCE hThisInstance, HINSTANCE hPrevInstance, LPSTR lpszArgument,
int nCmdShow) {
    HWND hwnd;
    MSG messages;
    WNDCLASSEX wincl; /*Define uma estrutura que contém informações sobre a classe da
janela. Esta estrutura é usada para registrar a classe da janela e definir algumas
propriedades importantes, como o estilo da janela, o ícone, o cursor e a função de
procedimento da janela.*/

    // Estrutura de definição da janela
    wincl.hInstance = hThisInstance;
    wincl.lpszClassName = szClassName;
    wincl.lpfnWndProc = WindowProcedure;
    wincl.style = CS_DBLCLKS;
    wincl.cbSize = sizeof(WNDCLASSEX);
    wincl.hIcon = LoadIcon(NULL, IDI_APPLICATION);
    wincl.hIconSm = LoadIcon(NULL, IDI_APPLICATION);
    wincl.hCursor = LoadCursor(NULL, IDC_ARROW);
    wincl.lpszMenuName = NULL;
    wincl.cbClsExtra = 0;
    wincl.cbWndExtra = 0;
    wincl.hbrBackground = (HBRUSH)COLOR_BACKGROUND;

    /*Registra a classe da janela com a API do Windows.
Se o registro falhar, o programa termina imediatamente.*/
    if (!RegisterClassEx(&wincl))
        return 0;

    // Criação da janela
    /*Cria uma janela usando a classe registrada.
Define o estilo da janela, o título, as dimensões e outros parâmetros
importantes.*/
    hwnd = CreateWindowEx(
        0,
        szClassName,
        "Clique do Mouse Detector",
        WS_OVERLAPPEDWINDOW,
        CW_USEDEFAULT,
        CW_USEDEFAULT,
        544,
        375,
        HWND_DESKTOP,
        NULL,
        hThisInstance,
        NULL
    );

    /*ShowWindow exibe a janela na tela em um estado especificado (nCmdShow).
    */
}
```

O próximo código detecta cliques no botão esquerdo do mouse e na tecla A, indicando no nome da janela se é um evento de teclado ou de mouse.

Vejamos:

```

c

#include

// Protótipo do procedimento de janela
LRESULT CALLBACK WindowProcedure(HWND, UINT, WPARAM, LPARAM);

// Nome da classe para a janela
char szClassName[] = "WindowClass";

int WINAPI WinMain(HINSTANCE hInstance, HINSTANCE hPrevInstance, LPSTR lpCmdLine, int
nCmdShow) {
    HWND hwnd;
    MSG messages;
    WNDCLASSEX wincl;

    // Estrutura de classe de janela
    wincl.hInstance = hInstance;
    wincl.lpszClassName = szClassName;
    wincl.lpfnWndProc = WindowProcedure;
    wincl.style = CS_DBLCLKS;
    wincl.cbSize = sizeof (WNDCLASSEX);

    // Ícones e ponteiros padrão
    wincl.hIcon = LoadIcon(NULL, IDI_APPLICATION);
    wincl.hIconSm = LoadIcon(NULL, IDI_APPLICATION);
    wincl.hCursor = LoadCursor(NULL, IDC_ARROW);
    wincl.lpszMenuName = NULL; // Sem menu
    wincl.cbClsExtra = 0;
    wincl.cbWndExtra = 0;
    wincl.hbrBackground = (HBRUSH) COLOR_BACKGROUND;

    if (!RegisterClassEx(&wincl))
        return 0;

    // Criação da janela
    hwnd = CreateWindowEx(
        0,
        szClassName,
        "Captura de Eventos de Teclado e Mouse",
        WS_OVERLAPPEDWINDOW,
        CW_USEDEFAULT,
        CW_USEDEFAULT,
        544,
        375,
        HWND_DESKTOP,
        NULL,
        hInstance,
        NULL
    );

    ShowWindow(hwnd, nCmdShow);
    UpdateWindow(hwnd);

    // Loop de mensagens
    while (GetMessage(&messages, NULL, 0, 0)) {
        TranslateMessage(&messages);
        DispatchMessage(&messages);
    }

    return messages.wParam;
}

// Função de callback para eventos da janela
LRESULT CALLBACK WindowProcedure(HWND hwnd, UINT message, WPARAM wParam, LPARAM lParam) {
    switch (message) {
        case WM_KEYDOWN:
            if (wParam == 'A') {
                MessageBox(hwnd, "Tecla A pressionada", "Evento de Teclado", MB_OK);
            }
            break;
    }
    return 0;
}

```



## Atividade 3

### Questão 1

Você está desenvolvendo um software educacional que requer interação frequente do usuário por meio do teclado e do mouse. Para implementar essa funcionalidade, você decide utilizar a biblioteca windows.h em um ambiente Dev-C++, no sistema operacional Windows. Considerando a função WindowProcedure da biblioteca windows.h, verifique a asserção e a razão a seguir e a relação entre elas.

Asserção: a função WindowProcedure é essencial para gerenciar eventos de teclado e mouse em um aplicativo Windows usando C.

Razão: a função WindowProcedure permite ao desenvolvedor definir o ícone e o cursor da aplicação.

Selecione a seguir a alternativa correta.

A

Ambas são verdadeiras, e a razão é a explicação correta da asserção.

B

Ambas são verdadeiras, mas a razão não é a explicação correta da asserção.

C

A asserção é verdadeira, mas a razão é falsa.

D

A asserção é falsa, mas a razão é verdadeira.

E

Ambas são falsas.



A alternativa B está correta.

A asserção é verdadeira, pois a função WindowProcedure realmente é importante para o manejo de eventos de teclado e mouse em aplicativos Windows. Essa função é encarregada de responder a eventos como pressionamentos de tecla e cliques de mouse. No entanto, a razão, embora verdadeira, não justifica a asserção.

A capacidade de definir ícones e cursores na aplicação é configurada na estrutura WNDCLASSEX e passada durante o registro da classe de janela, não sendo uma funcionalidade diretamente relacionada ao gerenciamento de eventos de entrada, que é o papel principal da WindowProcedure.

# Captura de eventos de teclado e mouse com OpenGL

A biblioteca GLUT facilita a criação de interfaces gráficas interativas em aplicações OpenGL, permitindo o manejo eficiente de eventos de teclado e mouse. Esse controle é essencial para desenvolver aplicações dinâmicas em C, desde jogos até softwares de visualização, proporcionando uma interação fluida e responsiva ao usuário.

Neste vídeo, enfatizaremos funções como `glutKeyboardFunc` e `glutSpecialFunc`, entre outras, demonstrando por meio de exemplos práticos como essas ferramentas podem enriquecer a interatividade em programas gráficos. Confira!



## Conteúdo interativo

Acesse a versão digital para assistir ao vídeo.

Para criarmos interações mais dinâmicas em aplicações, precisamos saber como gerenciar eventos de mouse e teclado. Assim, vamos usar as funções da GLUT, uma biblioteca que simplifica a criação de programas que utilizam OpenGL, para lidar com esses eventos em C.

Conheça as funções:

1

### `glutKeyboardFunc`

É útil para capturar a entrada do teclado em um programa OpenGL com GLUT. Você passa para ela um callback, que é basicamente uma função que será chamada sempre que uma tecla for pressionada.

2

### `glutSpecialFunc`

Funciona de maneira similar à `glutKeyboardFunc`, mas é usada para teclas especiais, como as setas do teclado. O callback recebe parâmetros: a tecla pressionada e as coordenadas x e y do mouse no momento do evento.

3

### `glutMouseFunc`

É utilizada para interações com o mouse. Essa função registra um callback que é chamado quando um evento de clique do mouse ocorre. No callback, você pode verificar qual botão foi pressionado e se ele foi clicado ou liberado, além da posição do mouse na tela.

4

### `glutMotionFunc` e `glutPassiveMotionFunc`

Representam as duas funções de GLUT para rastrear o movimento do mouse. A `glutMotionFunc` é chamada quando o mouse se move enquanto um botão está pressionado. Já a `glutPassiveMotionFunc` é usada para detectar movimentos do mouse quando nenhum botão está sendo pressionado.

## Atividade 4

### Questão 1

Em um projeto de desenvolvimento de um jogo 2D utilizando OpenGL e a biblioteca GLUT, um programador deseja implementar um sistema de controle em que o personagem principal se move de acordo com as teclas direcionais pressionadas. Considerando as funções do GLUT, qual delas deve ser usada para capturar os eventos de teclado especial e mover o personagem de acordo com o esperado?

A

glutKeyboardFunc

B

glutSpecialFunc

C

glutMouseFunc

D

glutMotionFunc

E

glutPassiveMotionFunc



A alternativa B está correta.

A função correta para capturar teclas especiais, como as setas direcionais do teclado, é `glutSpecialFunc`. Ela permite registrar um callback que é chamado sempre que uma tecla especial é pressionada.

## Demonstração prática da captura de eventos

Nesta atividade prática, exploraremos o processo criativo e técnico de desenvolver um jogo usando OpenGL — e, especificamente, a biblioteca GLUT para criar a interface gráfica. Criaremos uma versão simples do telejogo dos anos 1970, também conhecido como **Pong**. Esse jogo de tênis de mesa com gráficos bidimensionais oferece uma ótima introdução a conceitos essenciais, como captura de eventos de teclado e atualização de gráficos em movimento, fundamentais no desenvolvimento de jogos e outros softwares.

Veja neste vídeo como manipular as entradas do usuário e controlar os estados do Pong para oferecer uma experiência interativa aos usuários.



## Pong

O jogo consiste em duas pessoas controlando barras verticais em lados opostos da tela para rebater uma bola e evitar que ela passe por suas barras.



### Conteúdo interativo

Acesse a versão digital para assistir ao vídeo.

Vamos explorar como interagir com o jogo e entender cada parte do código.

## Como interagir com o jogo

- Pressione ENTER para iniciar ou reiniciar o jogo após alguém ganhar. Lembre-se: ele começa em estado de pausa.
- Use as teclas W (para cima) e S (para baixo) para mover o jogador 1.
- Utilize as teclas de seta ↑ (para cima) e ↓ (para baixo) para mover o jogador 2.
- Pressione ESC para sair do jogo.

## Explicação do código

### Configurações iniciais e variáveis globais

- Inicializam as constantes para largura e altura da janela, velocidade dos movimentos e outras configurações, como o número de rodadas.
- Definem variáveis para controlar a movimentação da bola, a direção de movimento e os pontos dos jogadores.

## Função main

- Configura e cria a janela do jogo e a subjanela do placar.
- Registra funções de callback para desenho na tela, eventos de teclado e temporizadores para movimentação dos jogadores e da bola.

## Funções de desenho (display e JanelaPlacar)

- É função do display desenhar as bordas do campo, os jogadores como linhas verticais e a bola como um ponto.
- É atribuição da JanelaPlacar mostrar o placar atual e as instruções e anuncia o vencedor quando um jogador alcançar o número necessário de pontos.

## Manipulação de eventos (keyboard e KeySpecial)

- É função do keyboard lidar com eventos de teclado para iniciar o jogo, mover o jogador 1 e sair do jogo.
- É papel de KeySpecial lidar com as teclas de seta para mover o jogador 2.

## Movimentação (moveJogador, jogador1, jogador2, bola)

- Atualizar a posição vertical do jogador com base em seu movimento atual é função de moveJogador.
- Jogador1 e jogador2 são funções temporizadas que chamam moveJogador para atualizar a posição de cada jogador.
- É característica de bola atualizar a sua posição, verificar colisões com os jogadores ou as bordas do campo e ajustar o placar.

## Utilidades (DesenhaTexto, DesenhaPtsPlacar, random e setPause)

- São características de DesenhaTexto e DesenhaPtsPlacar serem funções auxiliares para desenhar texto na tela.
- É função de random gerar um ângulo aleatório para o movimento da bola após colidir com um jogador.
- É atribuição de setPause reiniciar as posições e as configurações quando o jogo é pausado ou reiniciado.

Agora, vamos analisar o código:

```

c

#include
#include
#include
#include

#define WIDTH 600
#define HEIGHT 500
#define VELOCIDADE_BOLA 30
#define VELOCIDADE_JOGADORES 1
#define NUM_RODADAS 10

GLint movebolax=0;
GLint movebolay=0;
GLint direcao=1;
GLint direcao_vertical=1;
GLint move_jogador1Y=0;
GLint move_jogador2Y=0;

int TYJ1 = 0;
int TYJ2 = 0;

int janelaPlacar;
int janela;
static char label[200];
int pts_jogador1 = 0;
int pts_jogador2 = 0;

int movimento_jogador1 = 2;
int movimento_jogador2 = 2;

double angulo = 0.0;

bool pause = true;

void init(void);
void display(void);
void bola(int passo);
void keyboard(unsigned char key, int x, int y);
void KeySpecial(int key, int x, int y);
void DesenhaTexto(char *s);
void JanelaPlacar(void);
void moveJogador(int &move_jogadorY, int &movimento_jogador);
void jogador1(int passo);
void jogador2(int passo);
void displayAll(void);
int random(void);
void setPause(void);

int main(int argc, char** argv){
    glutInit(&argc, argv);

    glutInitDisplayMode(GLUT_DOUBLE | GLUT_RGB);
    glutInitWindowSize (WIDTH, HEIGHT);
    glutInitWindowPosition (100, 100);

    janela = glutCreateWindow ("Telejogo");
    glutIdleFunc(displayAll);
    init();
    glutDisplayFunc(display);

    glutTimerFunc(VELOCIDADE_JOGADORES,jogador1, 1);
    glutTimerFunc(VELOCIDADE_JOGADORES,jogador2, 1);

    glutTimerFunc(10,bola,1);

```

# Faça você mesmo!

## Questão 1

Imagine que você está desenvolvendo um jogo de tênis de mesa digital chamado Telejogo. Para aumentar a dificuldade e desafiar participantes, você decide alterar a velocidade da bola, pois isso é um crucial, porque pode tornar o jogo mais emocionante e competitivo.

Se quisermos aumentar a dificuldade do jogo alterando a velocidade da bola, qual das seguintes mudanças no código seria apropriada?

A

Aumentar o valor de VELOCIDADE\_BOLA de 30 para 50.

B

Diminuir o valor de VELOCIDADE\_BOLA de 30 para 10.

C

Aumentar o valor de VELOCIDADE\_JOGADORES de 1 para 5.

D

Diminuir o valor de NUM\_RODADAS de 10 para 5.

E

Aumentar o intervalo do temporizador da função bola para 20.



A alternativa B está correta.

O valor de VELOCIDADE\_BOLA no código do Telejogo controla o intervalo em milissegundos entre cada atualização da posição da bola. Reduzir esse intervalo aumenta a frequência de atualização, levando a bola a se mover com maior velocidade, o que, consequentemente, torna o jogo mais difícil. Portanto, diminuir o valor de 30 para 10 significa que a bola será atualizada mais frequentemente, aumentando sua velocidade no jogo.

## Considerações finais

### O que você aprendeu neste conteúdo?

- Linguagem C e controle detalhado sobre os recursos do computador.
- A estrutura de um programa em C para o desenvolvimento de software essencial.
- Prática de codificar em C e aplicação de conceitos em soluções reais.
- Utilização de bibliotecas para datas, horários e gráficos em C.
- Técnicas para usar OpenGL e GLUT na criação de gráficos avançados e interativos.
- Captura e resposta a eventos de teclado e mouse para desenvolver interfaces responsivas.

### Explore +

Quer aprender mais sobre esse assunto? Confira os artigos que selecionamos para você!

- Leia o artigo **Aplicação da biblioteca OpenGL em um jogo desenvolvido no C# com integração do Arduino Uno e Ionic Framework** e entenda como a integração de tecnologias de recursos necessários para a construção de um jogo de tiro ao alvo interage externamente com um circuito embarcado.
- Acesse o artigo **Realidade misturada: conceitos, ferramentas e aplicações** e veja como o autor propõe a combinação de cenas do mundo real com o virtual usando OpenGL como uma das ferramentas.
- Examine como os autores implementam um simulador de animações tridimensionais com OpenGL e Python no artigo **Interface para simulações da dinâmica de cabos com animação tridimensional**.

## Referências

COHEN, M.; MANSSOUR, I. H. **OpenGL: uma abordagem prática e objetiva**. São Paulo: Novatec Editora, 2006.

DEITEL, P. J.; DEITEL, H. M. **C: Como Programar**. 6. ed. São Paulo: Pearson, 2011.

HAWKINS, K.; ASTLE, D. **OpenGL game programming**. Boston: Cengage Learning, 2001.

KING, K. N. **C Programming: a modern approach**. 2. ed. Nova York: W. W. Norton & Company, 2008.

MANZANO, J. **Linguagem C acompanhada de uma xícara de café**. São Paulo: Érica, 2015.

SELLERS, G.; WRIGHT, R.; HAEMEL, N. **OpenGL superbible: comprehensive tutorial and reference**. 7. ed. Londres: Addison-Wesley Professional, 2015.