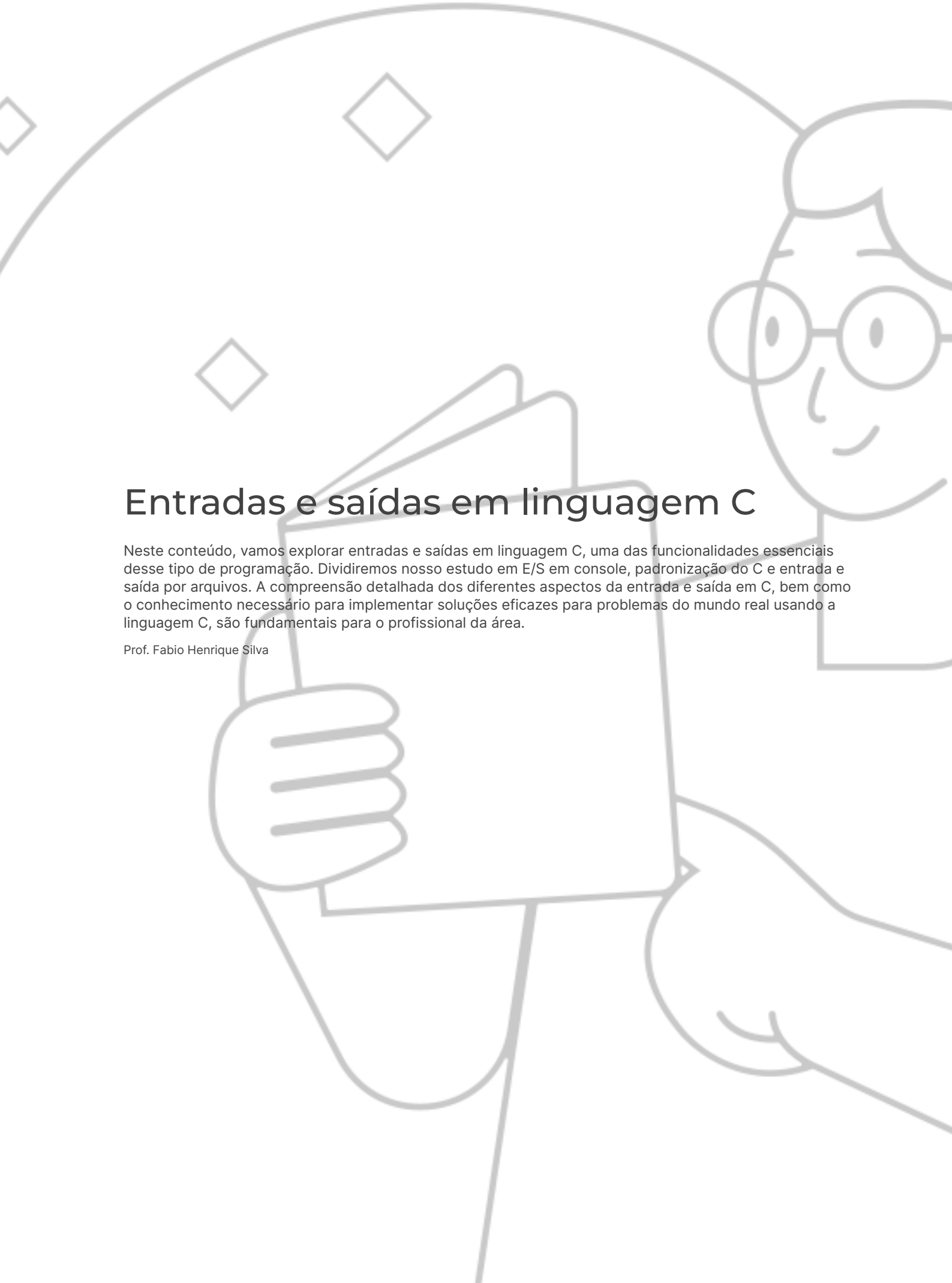


Entradas e saídas em linguagem C

Neste conteúdo, vamos explorar entradas e saídas em linguagem C, uma das funcionalidades essenciais desse tipo de programação. Dividiremos nosso estudo em E/S em console, padronização do C e entrada e saída por arquivos. A compreensão detalhada dos diferentes aspectos da entrada e saída em C, bem como o conhecimento necessário para implementar soluções eficazes para problemas do mundo real usando a linguagem C, são fundamentais para o profissional da área.

Prof. Fabio Henrique Silva

Preparação

O ambiente utilizado nas práticas foi pensado para o sistema operacional **Windows 10/11** com IDE Dev-C++. Mas você pode treinar em outros sistemas operacionais, como Linux ou Mac.

Objetivos

- Empregar as funções-padrão da linguagem C para entrada e saída pelo console.
- Reconhecer as particularidades dos padrões ANSI e UNIX em C.
- Aplicar as funções-padrão da linguagem C para entrada e saída por arquivos.

Introdução

Olá! Assista ao vídeo para conhecer os principais tópicos que serão abordados sobre entradas e saídas na linguagem C.

Conteúdo interativo

Acesse a versão digital para assistir ao vídeo.

Entrada (leitura) pelo console

Vamos explorar neste conteúdo a entrada de dados (input) em linguagem C, focando nas operações básicas de console sem manipulação de elementos gráficos. Abordaremos também funções como getchar() e getch(), além de alternativas seguras para leitura de strings, como fgets().

Para iniciar nosso estudo, confira neste vídeo o conceito fundamental de entrada pelo console em C e como os dados são capturados e manipulados diretamente pelo usuário.

Conteúdo interativo

Acesse a versão digital para assistir ao vídeo.

Lendo caracteres (input)

A linguagem C não possui palavras-chave específicas para entrada e saída (E/S), mas oferece um conjunto de funções robusto para essas operações. Aqui enfocaremos a E/S via console, deixando a E/S via arquivo para a próxima discussão.

Comentário

As funções de E/S padrão do ANSI C, que exploraremos aqui, são limitadas ao básico da entrada pelo teclado e da saída pela tela, sem manipulação de elementos gráficos. Isso varia de acordo com o ambiente de desenvolvimento.

Iniciando com as funcionalidades de entrada (leitura) de dados pelo console, vamos abordar a função **getchar()**, que é a forma mais básica de ler um caractere do teclado. Veja no exemplo a seguir que transformamos letras minúsculas em maiúsculas e vice-versa.

A seguir, teste o emulador de código. Insira alguns caracteres no campo Input, um por linha. Em seguida, clique em Executar e verifique a saída do código no campo Console.

Conteúdo interativo

esse a versão digital para executar o código.

A função getchar() possui um problema. Conforme definido pelo padrão ANSI C, ela armazena caracteres em um buffer até que o enter seja pressionado, o que pode limitar a interatividade em certos ambientes, como UNIX, em que o buffer de linha é uma prática comum.

Para uma leitura mais interativa, é possível optar por funções específicas do compilador que não estão no padrão ANSI C, como getch() no Windows. A biblioteca conio.h, exclusiva do Windows, inclui essa função, conforme demonstrado no exemplo a seguir.

```c
#include   // Inclui a biblioteca conio.h que contém funções específicas do console
#include   // Inclui a biblioteca padrão de entrada e saída
#include   // Inclui a biblioteca para funções de teste e mapeamento de caracteres

int main(void) {
    char caractere;  // Declara uma variável do tipo char para armazenar cada caractere
lido

    // Solicita ao usuário que digite um texto e informa como sair do programa
    printf("Entre com um texto (Digite ponto (.) para sair do programa): \n");
    do {
        caractere = getch();  // Lê um caractere do teclado sem ecoar na tela

        // Verifica se o caractere é uma letra minúscula
        if (islower(caractere)) {
            caractere = toupper(caractere);  // Converte para maiúscula se for minúscula
        }
        else {
            caractere = tolower(caractere);  // Converte para minúscula se não for
minúscula
        }

        putchar(caractere);  // Exibe o caractere convertido no console
    }
    while (caractere != '.');  // Repete o loop até que o caractere '.' seja digitado
}
```

Lendo strings

Temos a função gets()para a leitura de strings, mas ela é conhecida por seus problemas de segurança, como buffer overflow. Uma alternativa segura é usar fgets().

Digite uma string no campo de entrada e clique em Executar no emulador. Na sequência, observe a saída do código no campo de console.

Conteúdo interativo

esse a versão digital para executar o código.

Entrada formatada pelo console

Usamos scanf () para entrada formatada, pois pode manipular diversos tipos de dados. A função retorna o número de caracteres escritos e permite a inserção de diferentes formatos especificados por códigos precedidos por %. Por exemplo, %s para strings e %d para inteiros.

No emulador adiante, insira uma string no campo de entrada, clique em Executar e observe a saída do código no campo Console.

Conteúdo interativo

esse a versão digital para executar o código.

Atividade 1

Considerando que a linguagem C utiliza funções específicas para realizar a entrada de dados (input), como getchar() e getch(), essas funções se comportam de diferentes maneiras na forma como lidam com a entrada de caracteres.

A função getchar() lê caracteres do teclado e os armazena em um buffer até que o enter seja pressionado, enquanto getch() lê caracteres diretamente, sem ecoar na tela e sem armazenar em buffer de linha. Esses detalhes são importantes quando se deseja implementar programas que requerem interatividade imediata com o usuário.

Com base na descrição das funções getchar() e getch(), qual é a principal diferença no comportamento dessas funções que afeta a interatividade do usuário em ambientes como UNIX?

A

getchar() lê caracteres diretamente sem armazenar em buffer, enquanto getch() armazena em buffer até que o enter seja pressionado.

B

Ambas as funções leem caracteres sem armazenar em buffer de linha, promovendo máxima interatividade.

C

getchar() usa um buffer de linha, o que pode reduzir a interatividade, enquanto getch() faz a leitura dos caracteres de forma direta, sem exibi-los na tela nem utilizar um buffer de linha.

D

getch() lê caracteres e armazena-os em um buffer até que o enter seja pressionado, e getchar() não utiliza buffer de linha.

E

Ambas as funções ecoam caracteres na tela conforme são digitados pelo usuário.

A alternativa C está correta.

Enquanto getchar() utiliza um buffer de linha, aguardando o pressionamento da tecla enter para processar os dados, o que pode reduzir a interatividade, getch() lê caracteres diretamente sem ecoá-los na tela nem utilizar um buffer de linha, o que permite interação imediata. Portanto, A alternativa descreve com precisão a diferença no comportamento entre getchar() e getch() que influencia a interatividade do usuário.

Saída (escrita) pelo console

Neste conteúdo, abordaremos as funcionalidades de saída pelo console na linguagem de programação C, destacando o uso das funções putchar(), puts() e printf(), bem como as nuances de formatação e apresentação de dados. Vamos explicar os especificadores de formatos e os modificadores que permitem

ajustar a apresentação dos dados para saídas mais claras e formatadas, essenciais para uma visualização eficaz e profissional dos dados em aplicações desenvolvidas em C.

Para começar seus estudos no assunto, confira neste vídeo como realizar saídas pelo console em C e aprenda a manipular eficientemente a exibição de dados no terminal por meio de exemplos práticos e dicas úteis para programação em C.

Conteúdo interativo

Acesse a versão digital para assistir ao vídeo.

Veja como as funções putchar(), puts() e printf() podem ser usadas.

1

putchar()

É utilizada para escrever e exibir um único caractere no console. Sua simplicidade a torna ideal para situações em que é necessário imprimir caracteres individuais, como na construção de strings ou em laços que manipulam caracteres.

2

puts()

É útil para imprimir uma string seguida por uma nova linha e para exibir mensagens completas e linhas de texto de forma rápida e eficiente.

3

printf()

É a mais versátil e poderosa das funções de saída em C, permitindo a impressão de strings formatadas. Com ela, é possível inserir variáveis em meio a texto e controlar a formatação dos dados de saída.

Vamos explicar agora os especificadores de formatos e os modificadores que permitem ajustar a apresentação dos dados para saídas mais claras e formatadas.

Saída pelo console (output)

A **função puts()** é empregada para escrever uma string inteira, acrescentando automaticamente uma nova linha ao final. Essa característica torna puts() mais leve e rápida que printf(), especialmente quando a otimização é essencial. Veja o exemplo de uso de puts() no emulador a seguir e, após isso, clique em executar.

Conteúdo interativo

esse a versão digital para executar o código.

Diferentemente de puts(), **fputs()** permite a escrita em diferentes dispositivos de saída, como stdout ou arquivos.

Saída formatada pelo console

A função printf() é versátil, pois permite a formatação de diversos tipos de dados, como strings, caracteres e números. O protótipo da função é encontrado no cabeçalho STDIO.H. Veja!

```c
int printf(char *string_de_controle, lista_de_argumentos);
```

A partir disso, ela retorna o número de caracteres impressos ou um valor negativo em caso de erro. Os caracteres de controle dentro da string definem como os argumentos subsequentes serão formatados e exibidos. Vamos conferir um exemplo!

```c
printf("Eu gosto %s de %c", "muito", 'C');
```

A saída será: **"Eu gosto muito de C."**

Podemos utilizar os **caracteres especiais** em printf(), conforme a lista a seguir. Acompanhe!

Código	Conversão / formato do argumento
%d	Número decimal inteiro (int). Também pode ser usado **%i** como equivalente a %d.
%u	Número decimal natural (unsigned int), ou seja, sem sinal.
%o	Número inteiro representado na base octal. Exemplo: 41367 (corresponde ao decimal 17143).
%x	Número inteiro representado na base hexadecimal. Exemplo: 42f7 (corresponde ao decimal 17143). Se usarmos %X, as letras serão maiúsculas: 42F7.
%X	Hexadecimal com letras maiúsculas.
%f	Número decimal de ponto flutuante. No caso da função printf, devido às conversões implícitas da linguagem C, ele serve tanto para **float** como para **double**. No caso da função scanf, %f serve para **float,** e %lf serve para **double**.
%e	Número em notação científica, por exemplo 5.97e-12. Podemos usar %E para exibir o E maiúsculo (5.97E-12).
%E	Número em notação científica com o **e** maiúsculo.
%g	Escolhe automaticamente o mais apropriado entre %f e %e. Novamente, podemos usar %G para escolher entre %f e %E.
%p	Ponteiro: exibe em notação hexadecimal o endereço de memória do ponteiro.
%c	Caractere: imprime o caractere que tem o código ASCII correspondente ao valor dado.
%s	Sequência de caracteres (*string*, em inglês).
%%	Imprime um %

Tabela: Caracteres especiais em printf().

Wikibooks.org.

Consequentemente, para representar esses números, podemos utilizar:

%d

Para números inteiros com sinal.

%u

Para números sem sinal.

%f

Para números em ponto flutuante.

%e ou %E

Para números em notação científica.

Considerando isso, quando utilizamos %g ou %G, printf() escolhe automaticamente o formato mais curto e apropriado. Na sequência, veja um exemplo e clique em Executar no emulador.

Conteúdo interativo

esse a versão digital para executar o código.

Logo percebemos que os modificadores de formato ajustam a apresentação dos dados.

Exemplo

%05d garante que um número inteiro seja exibido com, pelo menos, cinco dígitos, preenchendo com zeros, se necessário. Para exibir dados em uma tabela com colunas alinhadas, você pode usar especificadores de largura e precisão.

Execute o código a seguir no emulador.

Conteúdo interativo

esse a versão digital para executar o código.

Por padrão, a saída é **justificada** à direita. Contudo, se você utilizar o sinal de menos (-) antes do especificador (como em %-10.2f), a saída será justificada à esquerda, facilitando a leitura em formatos tabulares. Confira!

```
c


#include

int main(void) {
    printf("justificado a direita: %8d\n", 100);
    printf("justificado a esquerda: %-8d\n", 100);
}
```

Essas ferramentas de formatação são essenciais para criar saídas legíveis e profissionalmente formatadas em aplicações C.

Atividade 2

No desenvolvimento de software usando a linguagem C, a saída de dados para o console é um recurso fundamental para a interação com o usuário e para a depuração de programas. Entre as funções disponíveis, putchar(), puts(), e printf() são amplamente utilizadas devido à sua eficácia e simplicidade.

A função putchar() é ideal para exibir um único caractere, enquanto puts() é mais eficiente para exibir strings, adicionando automaticamente uma nova linha ao final. Por outro lado, printf() oferece uma gama de opções de formatação para diversos tipos de dados, o que a torna extremamente versátil e poderosa para produzir saídas formatadas complexas, como números em formatos específicos, além de tabelas alinhadas e dados em notação científica.

Considerando as funções de saída de dados no console em C descritas anteriormente, qual delas seria a mais apropriada para gerar uma saída formatada que inclui strings, caracteres e números, permitindo especificar o formato de cada elemento incluído na saída?

A

A função putchar() seria a mais apropriada, pois permite uma saída detalhada de caracteres individuais.

B

A função puts() é a mais indicada, pois pode exibir strings inteiras e adicionar automaticamente uma nova linha no final.

C

A função printf() é a escolha correta, pois permite a formatação complexa de strings, caracteres e números, com controle detalhado sobre o formato de cada elemento.

D

A função fputs() deve ser usada, já que permite escrever em diferentes dispositivos de saída.

E

Todas as funções são igualmente apropriadas para produzir saídas formatadas complexas.

A alternativa C está correta.

A função printf() é extremamente versátil, pois possibilita especificar formatos de maneira detalhada para combinar texto literal com variáveis e especificadores de formato para uma variedade de tipos de dados. Essa funcionalidade torna printf() ideal para saídas formatadas complexas, permitindo controle preciso sobre a apresentação de strings, caracteres e números.

Demonstração prática de E/S pelo console

Nesta prática, exploraremos como a linguagem C permite a interação com o usuário por meio de entradas e saídas no console, utilizando, para isso, funções de alto nível. Compreender essas operações é fundamental para desenvolver aplicativos que requerem interação direta com o usuário.

Vamos nos concentrar nas funções scanf() para leitura de dados e printf() para exibição de informações, aprendendo como essas ferramentas podem ser aplicadas em situações do mundo real. Ao fim deste conteúdo, você será capaz de criar programas que respondem de maneira dinâmica às entradas do usuário e apresentam resultados de maneira clara e eficiente.

Mas primeiro, assista ao vídeo e ao guia completo para as operações de entrada e saída por meio do console. Veja como utilizar certas funções, a saber: scanf(), printf() e fgets(), para interagir de forma eficiente com o usuário no ambiente de programação em C.

Conteúdo interativo

Acesse a versão digital para assistir ao vídeo.

Roteiro de prática

Imagine que você está desenvolvendo um sistema para uma livraria que necessita de um programa para registrar novas aquisições de livros. Como o sistema deve solicitar ao usuário que insira o nome do livro, o autor e o número de páginas e, em seguida, confirmar a entrada, exibindo as informações de volta ao usuário?

Chave de resposta

- Utilizando as funções de alto nível scanf() e printf().
- O programa pedirá ao usuário que digite o nome do livro, o nome do autor e o número de páginas. Vamos captar esses dados com a função scanf().
- Após a entrada dos dados, com o uso da função printf( ), o programa exibirá as informações inseridas de volta ao usuário, para confirmá-las. Esse processo ajuda a garantir que as informações digitadas estão exatas e completas, facilitando a gestão eficaz das novas aquisições da livraria.
- A solução para o problema demonstra um uso das funções de entrada e saída em C que é prático, relevante e adequado para aplicações no mundo real, as quais requerem interação direta com o usuário.

Agora, veja o código com a solução:

```c
#include

int main() {
   char livro[100], autor[100];
   int paginas;

   printf("Digite o nome do livro: ");
   scanf("%99s", livro);  // Lê uma string até o espaço
   printf("Digite o nome do autor: ");
   scanf("%99s", autor);  // Lê uma string até o espaço
   printf("Digite o número de páginas: ");
   scanf("%d", &paginas);

   printf("\nVocê registrou o livro: '%s' de %s, com %d páginas.\n", livro, autor,
paginas);
   return 0;
}
```

Faça você mesmo!

Suponha que um sistema semelhante ao da livraria está em uso, mas os usuários reclamam que não conseguem inserir nomes de livros ou autores com espaços.

Qual das seguintes modificações você deveria realizar no código para permitir a leitura de strings com espaços?

A

Substituir %99s por %99[^\n] no scanf para livro e autor.

B

Aumentar o tamanho do array para 200 caracteres.

C

Usar gets() em vez de scanf().

D

Dividir o input em muitas variáveis.

E

Nenhuma alteração é necessária.

A alternativa A está correta.

Para permitir que os usuários insiram nomes de livros ou autores que contenham espaços, você deve substituir %99s por %99[^\n] no scanf, pois isso permite a leitura de uma linha inteira até encontrar uma quebra de linha, incluindo espaços, adequando-se à necessidade de inserir nomes completos de livros e autores.

Confira como fazer a inclusão:

```
c

#include



int main() {

    char livro[100], autor[100];

   int paginas;



   printf("Digite o nome do livro: ");

   scanf(" %99[^\n]", livro);  // Modificação para ler uma string com espaços

   printf("Digite o nome do autor: ");

   scanf(" %99[^\n]", autor);  // Modificação para ler uma string com espaços

   printf("Digite o numero de paginas: ");

   scanf("%d", &paginas);



   printf("\nVoce registrou o livro: '%s' de %s, com %d paginas.\n", livro, autor,
paginas);

   return 0;

}
```

O padrão ANSI/ISO C

É importante abordar a evolução da linguagem de programação C, desde o projeto Multics até o desenvolvimento dos padrões ANSI, que aprimoram sua portabilidade e eficiência.

O C experimentou a transição de uma programação inicial em linguagem de montagem para outra estruturada e amplamente adotada, o que destaca sua influência duradoura e seu sucesso no cenário da programação, o qual se apoia em padrões que facilitam sua utilização em diferentes plataformas e sistemas operacionais.

Para iniciar seus estudos, acompanhe neste vídeo os fundamentos essenciais da linguagem C e as práticas recomendadas para desenvolver aplicações robustas e eficientes em C.

Conteúdo interativo

Acesse a versão digital para assistir ao vídeo.

O que é o padrão ANSI/ISO C?

ANSI C, ISO C e Standard C são um padrão para a linguagem de programação C publicado pelo American National Standards Institute (ANSI), pela ISO/IEC JTC 1/SC 22/WG 14, da Organização Internacional de Padronização (ISO) e pela Comissão Eletrotécnica Internacional (IEC).

> O termo ANSI C se refere especificamente à versão original e mais bem suportada do padrão (conhecido como C89 ou C90). Os desenvolvedores que utilizam a linguagem C são encorajados a se adequarem aos padrões, pois isso ajuda a portabilidade entre compiladores.

Atualmente, quase todos os compiladores de amplo uso suportam o ANSI C. Códigos fonte que aderem estritamente ao padrão C, mas não fazem suposições baseadas em hardware específico, têm alta probabilidade de serem compilados corretamente em qualquer plataforma que possua implementação C compatível.

Na ausência dessas precauções, a maioria dos programas tende a ser compilável somente em plataformas específicas ou com compiladores particulares, devido, por exemplo, ao uso de bibliotecas fora do padrão ou dependências relacionadas a características únicas do compilador ou plataforma.

Exemplo

Bibliotecas de interface gráfica e compilador ou plataforma com determinados tipos de dados com tamanho específico.

O termo **C estrito** se refere a uma configuração específica do compilador, que assegura a aderência rigorosa ao padrão C. Um compilador que opera em modo estrito que ele segue o padrão C sem desvios ou extensões.

A linguagem C, em sua essência, é bastante enxuta, e tudo o que não faz parte de seu núcleo é oferecido por meio de bibliotecas. Por exemplo, a biblioteca stdio (descrita, por exemplo, na seção 7.19 da norma ISO/IEC 9899:1999), representa a biblioteca padrão de entrada e saída, responsável por fornecer funções como **printf()**.

Existem bibliotecas que são desenvolvidas por programadores comuns e disponibilizadas para uso geral, porque não seguem o padrão C. Dessa forma, podemos entender a GNU C de duas maneiras distintas. Veja!

Primeira

Pode se referir ao compilador de C, que é parte da GNU Compiler Collection (GCC).

Segunda

Pode denotar a configuração padrão do compilador GCC, que não adere estritamente aos padrões da linguagem C, conhecida como GNU C não padrão.

Um exemplo notável do uso dessa configuração é o kernel, do Linux, que é desenvolvido utilizando as extensões GNU C, e não o padrão C puro.

Para garantir que um programa seja compilado de acordo com um padrão específico de C, como o C99 ou o C17, é necessário especificar isso na linha de comando do compilador.

As origens do C

O percurso do C começou com o tropeço do projeto Multics, em 1969, uma tentativa da General Electric, do Massachusetts Institute of Technology (MIT) e da Bell Laboratories de criar um sistema operacional avançado. O Multics não cumpriu as expectativas devido à sua complexidade para a época. No entanto, suas falhas abriram caminho para o desenvolvimento da linguagem C.

Ken Thompson, que havia trabalhado no projeto, estava determinado a desenvolver outro sistema operacional. Junto com Dennis Ritchie, começaram a desenvolver um sistema operacional simplificado para o PDP-7, o UNIX, que antecedeu o C.

Como programar em assembler era, e ainda é, considerado complicado, Thompson desenvolveu a linguagem B, que não foi bem-sucedida devido às restrições de memória. Mas com o advento do PDP-11, ele reescreveu o sistema operacional em assembler do PDP-11.

Ken Thompson e Dennis Ritchie, os pioneiros do UNIX, precursor do C.

Dennis Ritchie, por sua vez, aprimorou o B no PDP-11, criando o Novo B, que rapidamente evoluiu para o C, que era compilado e tinha um sistema robusto de tipos.

Na década de 1970, o C já era reconhecido e refinado, ajustando funcionalidades e estendendo tipos de dados para novos hardwares. Em 1978, o compilador portátil de C (pcc) foi desenvolvido por Steve Johnson, e a influência do C continuou a crescer com a publicação da *The C Programming Language*, por Brian Kernighan e Dennis Ritchie. A adoção generalizada dessa linguagem, conhecida como K&R C, demonstrou sua eficácia e popularidade.

Curiosidade

A primeira versão do C, embora relevante, tinha limitações que complicavam o desenvolvimento e a portabilidade de programas. Para resolver isso, o ANSI interveio em 1983, formando o comitê X3J11 para estabelecer um padrão para a linguagem C, que foi aprovado em 1989.

O padrão ANSI C melhorou a eficiência e a portabilidade do C por meio da clarificação de tipos de dados e da definição de funções de biblioteca padrão, além de um sistema de arquivos independente do sistema operacional.

Evolução histórica do padrão ANSI C e ISO C

Confira a seguir a trajetória da linguagem C, que é marcada por várias fases de padronização ao longo do tempo:

1972-1989

K&R C
Desenvolvimento da linguagem C por Dennis Ritchie na Bell Labs, o que representou um avanço significativo na programação de sistemas. Conhecida como K&R C, em referência ao livro *The C Programming Language*, escrito por Brian Kernighan e o próprio Dennis Ritchie, essa versão estabeleceu os fundamentos da linguagem e predominou como padrão até o primeiro padrão oficial ser estabelecido em 1989 pelo ANSI.

1989

C89 (ANSI C)
Estabelecimento do primeiro padrão oficial de C pelo ANSI, conhecido como C89 ou ANSI C. Desde essa época, o American National Standards Institute deixou de ter papel direto na evolução do C, passando a ser apenas uma das várias entidades envolvidas no padrão ISO.

1990

C90
Adoção internacional do padrão C89 pela ISO, formalmente reconhecido como C90. Tecnicamente, ele é idêntico ao C89.

1995

C95
Introdução de uma pequena atualização, conhecida como C95, que incluiu suporte ampliado para caracteres.

1999

C99
Revisão substancial do padrão C, resultando no C99, que introduziu suporte a múltiplos processamentos e novas capacidades de *threading*. Permaneceu como o padrão até 2011.

2011

C11
Lançamento do padrão C11, trazendo melhorias significativas, incluindo suporte a *multi-threading* e outras funcionalidades modernas.

2017/2018

C17/C18
Conclusão da versão mais recente da linguagem C, informalmente conhecida como C17 ou C18, mas lançada como ISO 9899:2018. Ela enfocou correções de diversos erros, mas sem introduzir novos recursos significativos.

A contínua evolução e padronização da linguagem C ao longo das décadas é refletida por diversos marcos. A linguagem C se adapta às necessidades crescentes de desenvolvimento de software e mantém sua relevância na computação moderna.

Atividade 1

A evolução da linguagem de programação C, desde sua concepção até os padrões modernos, é um exemplo fascinante de desenvolvimento tecnológico e colaboração. A jornada começou nos anos 1960 e 1970, quando Ken Thompson e Dennis Ritchie criaram a linguagem, inicialmente para uso no sistema operacional UNIX, no PDP-7. Esse desenvolvimento inicial, embora promissor, logo encontrou limitações que necessitavam de melhorias significativas, culminando na criação dos padrões ANSI e ISO para garantir a portabilidade e eficiência.

Qual foi o impacto significativo do padrão ANSI C, estabelecido em 1989, na evolução da linguagem de programação C?

A

O padrão ANSI C introduziu a programação orientada a objetos, o que revolucionou a escrita de software em C.

B

O padrão ANSI C, aprovado em 1989, melhorou a eficiência e a portabilidade do C por meio da clarificação de tipos de dados e das definições de funções de biblioteca.

C

ANSI C reduziu a popularidade da linguagem C ao impor restrições rígidas que limitaram a criatividade dos programadores.

D

ANSI C foi responsável pela adição de recursos de processamento gráfico avançado, alinhando C com linguagens, como Python e Java.

E

O padrão estabelecido em 1989 removeu características essenciais, como ponteiros e gerenciamento manual de memória, para simplificar a linguagem.

A alternativa B está correta.

O padrão ANSI C, estabelecido em 1989, foi um marco significativo na evolução da linguagem de programação C. A padronização realizada pelo ANSI trouxe várias melhorias que impactaram profundamente a forma como a linguagem era utilizada e desenvolvida.

O padrão UNIX

Reconhecido por sua simplicidade, ferramenta de linha de comando e modularidade, o padrão UNIX (ou *Uniplexed information and computing)* influenciou profundamente o desenvolvimento de sistemas operacionais.

Curiosidade

O UNIX foi originalmente escrito em assembly e posteriormente em C, tornando-se o primeiro sistema operacional escrito nessa linguagem.

Uma das evoluções significativas no uso do C foi a introdução do padrão POSIX (ou *Portable operating system interface*), estabelecido inicialmente em 1988. Esse padrão desempenha importante papel na uniformização das chamadas de sistema entre diferentes variantes de sistemas operacionais Unix-like, facilitando a portabilidade dos programas em múltiplos ambientes e sua funcionalidade.

Ao explorarmos como o POSIX estende as capacidades do ANSI-C, ganhamos uma compreensão mais aprofundada sobre como as interfaces de programação podem influenciar a compatibilidade e a eficiência das aplicações modernas.

Assista ao vídeo e entenda as particularidades de programar em C no ambiente UNIX.

Conteúdo interativo

Acesse a versão digital para assistir ao vídeo.

O padrão UNIX para programação em C

A programação de sistema se refere ao desenvolvimento de programas que oferecem serviços essenciais de um sistema operacional, que inclui:

- Interpretadores de linha de comando.
- Funções de entrada e saída de alto nível (como as funções de stream I/O em C).
- Compiladores de linguagem.
- Carregadores.
- Sistemas operacionais.

Apesar das características que possui, os sistemas não se limitam a elas. Dessa forma, notamos que esses componentes são fundamentais para o funcionamento de um sistema computacional e a sua manutenção.

Um exemplo dessa prática pode ser visto em ambientes UNIX, nos quais diferentes shells operam como programas em C para fornecer interativamente serviços do kernel no nível do usuário, permitindo até mesmo a emulação de outros sistemas operacionais através de suas linguagens interativas. Veja!

1

Bourne-shell (sh)

É um interpretador de linha de comando padrão nos sistemas UNIX e UNIX-like. Foi desenvolvido por Stephen Bourne na década de 1970 e é conhecido por ser robusto e eficiente, sendo amplamente utilizado para automação de tarefas e scripting no ambiente UNIX.

2

C Shell (csh)

Trata-se de outro interpretador de linha de comando para sistemas UNIX, criado por Bill Joy em meados da década de 1970. Ele oferece uma sintaxe semelhante à linguagem C e inclui recursos adicionais, como histórico de comandos e expansão de variáveis, que o tornam popular entre alguns usuários UNIX.

3 Outros shells operam como programas em C

Existem outros interpretadores de linha de comando para UNIX, que são implementados em C. Esses shells operam em níveis variados de interatividade e funcionalidade, mas todos são fundamentais para a execução de comandos, a manipulação de arquivos e a interação com o sistema operacional no nível do usuário.

Os shells executam comandos e podem ser estendidos e personalizados para oferecer funcionalidades adicionais, como scripts complexos, automação de processos e até mesmo a emulação de ambientes de outros sistemas operacionais por meio de capacidades interativas e scripting.

O núcleo do sistema operacional, ou kernel, encapsula os recursos internos essenciais que servem de base para as aplicações utilizadas pelos usuários. É possível acessar o kernel por meio das chamadas de sistema (*system calls*), que permitem aos programas acessarem e manipularem funções fundamentais do sistema operacional de forma eficiente e segura.

As chamadas de sistema variam entre os diferentes sistemas operacionais, mas os conceitos fundamentais geralmente são consistentes.

> Em sistemas UNIX, as interfaces de chamadas de sistema são definidas em linguagem C. Cada chamada de sistema do UNIX geralmente possui uma função homônima em uma biblioteca C. As chamadas de sistema podem ser consideradas funções em C.

A interface de chamada de sistema é documentada na seção 2 do manual do programador UNIX. A seção 3 descreve funções de uso geral que estão disponíveis para os programadores. Elas não são pontos de entrada no kernel, mas podem fazer uso de chamadas de sistema.

Exemplo

A função printf pode utilizar a chamada write para processar saídas, enquanto a função atoi não utiliza chamadas de sistema.

Durante a década de 1980, a diversificação das versões do UNIX e suas diferenças criaram demandas por padronização, resultando em padrões como ANSI C e IEEE POSIX.

POSIX

Além de apoiar a portabilidade de aplicações no nível do código fonte, o C continua a fornecer uma interface de sistema operacional como uma opção, tal qual a interface Unix no K&R C.

Esse recurso opcional, bem como o padrão POSIX, lançado originalmente como IEEE Std. 1003.1-1988, é chamado de Interface de programação de aplicações unix-like (API), que estende a facilidade do ANSI-C para fornecer um ambiente de programação C mais rico sob qualquer sistema operacional.

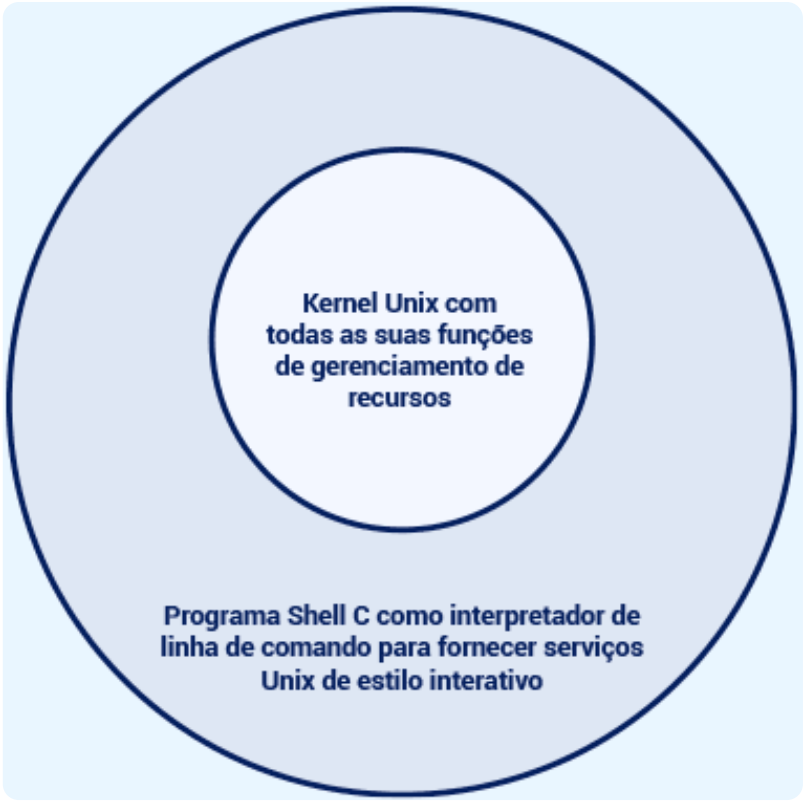
Curiosidade

O POSIX, conhecido como portable operating system interface for unix, visa padronizar as chamadas de sistema e unificar as várias versões do UNIX. Isso significa que aplicativos desenvolvidos conforme o padrão POSIX têm potencial para serem executados em qualquer sistema operacional que ofereça suporte a esse padrão.

Atualmente, a maioria dos sistemas operacionais incorpora o POSIX. Em comparação, o Windows utiliza seu próprio conjunto de chamadas de sistema, chamado WinAPI, que, ao contrário do POSIX, pode incluir incrementos ou variações em cada nova versão lançada pelo Windows.

O ANSI-C estendido não pode ser tratado como uma simples linguagem de aplicação (como Basic, FORTRAN ou Pascal), porque representa uma máquina virtual poderosa. Por exemplo, essa máquina virtual é capaz de executar instruções em C e fornecer instruções virtuais para acessar os recursos do sistema operacional de suporte, como mostrado nas imagens a seguir.

Representação da máquina virtual C.



Representação da máquina virtual interpretativa de Shell.

Atividade 2

A linguagem de programação C desempenha papel crítico em sistemas operacionais, proporcionando uma interface padrão que aumenta a portabilidade e a funcionalidade em diversos ambientes computacionais. Com a adoção do padrão POSIX, um marco em 1988, a linguagem C estendeu sua aplicabilidade, permitindo uma interface de programação mais rica, que abrange várias plataformas de sistema operacional. Esse padrão busca unificar e facilitar o desenvolvimento de software em diferentes sistemas Unix-like, estendendo a flexibilidade do ANSI-C.

Qual é o papel do padrão POSIX na interface de programação de sistemas operacionais?

A

O padrão POSIX permite a criação exclusiva de interfaces gráficas de usuário, diferenciando-o completamente de outras APIs, como WinAPI.

B

O POSIX foca primariamente aumentar a segurança dos sistemas operacionais, restringindo o acesso ao nível do kernel por meio de chamadas de sistema padronizadas.

C

A principal função do POSIX é substituir completamente o ANSI-C, estabelecendo um novo padrão para todas as linguagens de programação, e não apenas o C.

D

O POSIX elimina a necessidade de usar a linguagem C para programação de sistemas, promovendo outras linguagens mais modernas, como Python e Java.

E

POSIX padroniza chamadas de sistema para unificar diversas versões do Unix, permitindo que aplicações escritas para um sistema possam ser executadas em outros com suporte POSIX.

A alternativa E está correta.

O padrão POSIX foi criado para uniformizar as interfaces de sistemas operacionais, especificamente as chamadas de sistema, a fim de garantir que programas desenvolvidos para um ambiente Unix-like sejam compatíveis com outros que também implementaram esse padrão.

E/S ANSI/ISO C ou UNIX C: qual adotar?

Enquanto o ANSI/ISO C oferece uma base padronizada e amplamente suportada para a portabilidade entre diferentes sistemas operacionais, o UNIX C proporciona acesso a funcionalidades específicas e otimizadas dos sistemas Unix-like. A decisão de qual abordagem adotar depende de fatores como:

- O ambiente alvo.
- Os requisitos específicos de desempenho.
- A necessidade de conformidade com padrões internacionais.

Para iniciar seus estudos sobre o assunto, confira neste vídeo exemplos práticos de situações em que os padrões ANSI/ISO C e UNIX C podem ser aplicados.

Conteúdo interativo

Acesse a versão digital para assistir ao vídeo.

Entendendo as diferenças entre E/S ANSI/ISO C e UNIX C

C tem uma vantagem distinta no acesso às funções do sistema operacional UNIX, pois quase todas as funções do kernel Unix são implementadas em C. Assim, você pode utilizar a linguagem para interação com o kernel Unix, constituindo uma ferramenta ideal para a manutenção do UNIX.

Vamos exemplificar para o caso da programação de entrada e saída – E/S. No sistema Unix, as chamadas de sistema de entrada/saída (E/S) formam a base para manipular arquivos, e até as funções de E/S de fluxo, definidas pelas normas ANSI-I/O, dependem dessas chamadas de sistema para realizar suas operações.

Atenção

Ao usarmos a função fopen, na verdade, utilizamos a chamada de sistema open. Da mesma forma, getc utiliza indiretamente a read, putc usa a write, e fclose, a close. A principal diferença entre essas abordagens de E/S está na presença de um buffer intermediário na E/S de fluxo, que gerencia o fluxo de dados em diferentes níveis de abstração, como caracteres individuais, strings e dados formatados.

Você pode optar pelo uso de E/S de baixo nível, caso prefira ou necessite desenvolver aplicações que façam referências às chamadas de sistema. Frequentemente, a E/S de baixo nível é preferida em utilitários de sistema operacional, no qual o desempenho é crítico em detrimento da E/S de fluxo (stream I/O) ou E/S de alto nível.

Considerando isso, acompanhe como empregar as abordagens da E/S de alto nível e da E/S de baixo nível para escrever em um arquivo. Vamos lá!

Abordagem de alto nível

Utilizamos a função **fprintf()** para escrever dados formatados em um arquivo.

```
c


FILE *file = fopen("exemplo.txt", "w"); // Aberto para escrita
fprintf(file, "Hello, %s!\n", "World");
```

Abordagem de baixo nível

Usamos a função **write()** para escrever em um arquivo.

```
c


int fd = open("exemplo.txt", O_CREAT | O_WRONLY | O_TRUNC, 0644);
const char *text = "Hello, World!\n";
ssize_t bytes_written = write(fd, text, strlen(text));
```

Na abordagem de baixo nível, você tem mais controle sobre o processo de escrita, podendo manipular diretamente os dados binários.

Além disso, você pode modificar os vários modos/flags da operação de gravação para obter os resultados desejados, como por exemplo o I/O direto (O_DIRECT).

Diante disso, vamos conferir uma diferença fundamental entre a E/S de baixo nível e a E/S tradicional.

Na E/S de baixo nível

Podemos habilitar a E/S direta, que não utiliza o Page Cache do kernel Linux e fornece acesso direto à E/S subjacente, proporcionando operações de entrada e saída de baixo nível.

Na E/S tradicional

Reserva-se certa quantidade de memória do sistema para armazenar em cache os acessos ao disco do sistema de arquivos, o kernel acaba otimizando o acesso a eles. A abordagem de cache do Linux é chamada de write-back.

Em se tratando de **write-back**, se os dados forem gravados no disco, eles serão armazenados na memória do cache e marcados como sujos lá até que sejam sincronizados com o disco. Em outras palavras: todos os dados que você tentar gravar no disco serão armazenados primeiro no cache da página. Dá para imaginar que isso pode implicar na velocidade de leitura ou gravação.

No entanto, ao usar a abordagem de baixo nível, é possível que você precise gerenciar o buffering e lidar com a codificação de texto, se estiver trabalhando com arquivos de texto.

Atividade 3

A linguagem C é notória por seu papel integral no desenvolvimento do sistema operacional UNIX e em seu e funcionamento, proporcionando uma interface direta e eficiente para manipulação de recursos do sistema. Desde os anos 1980, a linguagem tem sido amplamente utilizada para adicionar novos serviços ao kernel Unix, refletindo sua capacidade de lidar com operações críticas, como gerenciamento de recursos e suporte à rede. A interação entre o padrão ANSI C e as especificações POSIX permitiu que o C oferecesse interfaces de programação compatíveis com UNIX em qualquer ambiente que suporte POSIX.

Como o padrão POSIX influenciou a capacidade do C de interagir com funções do sistema operacional UNIX?

A

A inclusão das especificações POSIX no padrão C permitiu que ele oferecesse uma interface semelhante à UNIX como serviço opcional, aumentando sua portabilidade entre diferentes sistemas operacionais.

B

O padrão POSIX permitiu a criação de novas chamadas de sistema específicas para C, que antes eram incompatíveis com outros sistemas operacionais além do Unix.

C

Ao alinhar-se ao POSIX, o C foi limitado em suas capacidades de interação com o sistema operacional, sendo restrito apenas a funções não essenciais do kernel Unix.

D

O POSIX restringiu o uso de C para aplicações de baixo nível, como a manipulação direta de hardware, o que reduziu sua relevância em sistemas operacionais modernos.

E

Com o POSIX, o C se tornou obsoleto, sendo substituído por linguagens de programação mais modernas que oferecem melhor suporte a interfaces gráficas e multimídia.

A alternativa A está correta.

A inclusão das especificações POSIX no padrão C teve impacto significativo na portabilidade da linguagem e em sua utilidade. O padrão POSIX (*portable operating system interface*) foi desenvolvido para garantir a compatibilidade entre diferentes sistemas operacionais que se baseiam em UNIX, proporcionando uma base comum para a programação de sistemas.

Fundamentos do sistema de arquivos

Veja como manipular arquivos usando um programa em C, abordando desde a criação até a modificação de arquivos binários e de texto, com exemplos de funções como:

fopen()

Usada para abrir arquivos. Ela retorna um ponteiro para o arquivo que foi aberto, ou NULL, se houver algum erro na abertura.

fclose()

Utilizada para fechar um arquivo que foi previamente aberto com fopen().

Vamos detalhar os tipos de arquivos suportados e as operações que podem ser realizadas com eles, oferecendo uma base sólida para o gerenciamento de dados em C.

Iniciando nosso estudo sobre o assunto, acompanhe neste vídeo os fundamentos essenciais dos sistemas de arquivos, explorando como os dados são organizados, armazenados, recuperados e gerenciados em sistemas de computação.

Conteúdo interativo

Acesse a versão digital para assistir ao vídeo.

Conceitos

Um arquivo é uma unidade lógica de dados criado por processos. Um sistema de arquivos (ou FS, do inglês *file system*) é responsável por organizar e facilitar o acesso a eles, funcionando como um serviço de armazenamento de dados que permite compartilhar o armazenamento em massa entre aplicativos.

Os sistemas de arquivos de disco aproveitam a capacidade dos dispositivos de armazenamento para acessarem dados de forma aleatória e rápida, possibilitando que múltiplos usuários ou processos acessem diferentes conjuntos de dados sem que a ordem sequencial de armazenamento impacte o desempenho.

Na sequência, confira alguns exemplos de sistemas de arquivos.

FAT
É um sistema de arquivos desenvolvido pela Microsoft e inicialmente para disquetes, mas depois ele foi adotado para unidades de disco rígido e outros dispositivos de armazenamento.

exFAT

É uma extensão do sistema de arquivos FAT, também desenvolvido pela Microsoft, para suportar dispositivos de armazenamento maiores e mais modernos, como dispositivos USB e cartões de memória.

NTFS

Trata-se do sistema de arquivos padrão para todas as versões do Windows, desde o Windows NT. Suporta tamanhos de arquivo muito grandes, segurança avançada de dados, permissões de arquivo e diretório, bem como compressão de arquivos e criptografia.

ext4

É o sistema de arquivos padrão para muitas distribuições Linux modernas. Foi projetado como uma evolução do sistema de arquivos ext3, oferecendo melhor desempenho e escalabilidade para grandes volumes de dados.

Também existem sistemas de arquivos direcionados para outros tipos de hardware, que armazenam arquivos, como discos óticos e dispositivos de armazenamento em rede.

Graças a um conjunto de funções oferecido pelo sistema de entrada/saída (E/S) de arquivos em C, um programa em C pode realizar diversas operações em arquivos no seu computador, como:

- Ler
- Escrever
- Mover
- Criar

O C oferece uma variedade de funções para gerenciar arquivos, incluindo criação, abertura, leitura, escrita e fechamento. Esses arquivos são fundamentais para armazenar dados relevantes e são manipulados em C para acessar e modificar esses dados.

Existem, principalmente, dois tipos de arquivos que podem ser manipulados. Veja!

Arquivos de texto

São arquivos simples e salvos com a extensão .txt, que qualquer editor de texto pode criar ou modificar. Armazenam dados em caracteres ASCII e são usados principalmente para guardar textos.

Arquivos binários

Armazenam dados em formato binário, em vez de caracteres ASCII, e são usados principalmente para informações numéricas como inteiros (int), pontos flutuantes (float) e dupla precisão (double). Os dados são armazenados como uma sequência de 0s e 1s.

Operações com arquivos em C

A linguagem oferece um conjunto robusto de funções para realizar várias operações de entrada/saída (E/S) em arquivos. Veja a seguir as operações mais importantes que você pode realizar com arquivos em C.

a, a+, w ou w+
Cria um novo arquivo.

fopen
Abre um arquivo existente.

fscanf ou fgets
Lê um arquivo.

fprintf ou fputs
Escreve em um arquivo.

fseek ou rewind
Move o ponto de vista dentro do arquivo para uma posição específica.

fclose
Fecha um arquivo.

Para trabalhar com arquivos, usamos um ponteiro especial (ponteiro de arquivo), que pode ser declarado da seguinte forma:

```
c


FILE *file_ptr;
file_ptr = fopen("fileName.txt", "w");
```

Modos de arquivo em C

Ao trabalhar com arquivos em C, é importante selecionar o modo de abertura adequado para suas necessidades específicas. Ele determina como o arquivo será manipulado e quais operações serão permitidas.

O modo de abertura do arquivo w, no exemplo anterior, pode variar conforme a necessidade. Vamos conferir!

r

Abre o arquivo para leitura. Se o arquivo não puder ser aberto, retorna NULL.

rb

Abre para leitura em modo binário. Retorna NULL se o arquivo não existir.

w

Cria ou sobrescreve um arquivo existente. Retorna NULL se não puder ser aberto.

wb

Abre para escrita em modo binário. Cria o arquivo se não existir.

a

Disponibiliza o arquivo para adicionar conteúdo no final. Cria um novo arquivo se não existir.

ab

Abre para adicionar em modo binário.

r+, rb+, w+, wb+, a+ e ab+

Trata-se de combinações dos modos anteriores para ler, escrever, adicionar e trabalhar em modo binário ou normal.

Se o seu objetivo é manipular arquivos binários, você usará principalmente modos como rb, wb, ab, rb+, wb+ e ab+, entre outros.

Atividade 1

Em um curso de programação em C, um estudante está usando essa linguagem para aprender sobre a manipulação de arquivos. Durante uma sessão de laboratório, ele precisa criar um programa que lide com arquivos binários para armazenar e recuperar dados numéricos. O aluno entende os fundamentos do sistema de arquivos e sabe que precisa escolher o modo correto de abertura para completar sua tarefa com sucesso.

Considerando que o estudante pretende ler e escrever dados em um arquivo binário existente, qual dos modos de abertura de arquivo ele deve utilizar para abrir o arquivo corretamente?

A

r

B

rb

C

w+

D

rb+

E

ab

A alternativa D está correta.

rb+ é o modo de abertura de arquivo que cumpre todos os requisitos especificados na questão: permite a leitura e a escrita de dados em um arquivo binário existente, sem apagar o conteúdo atual do arquivo.

Vamos analisar cada modo de abertura para entender por que rb+ é o mais adequado.

1. **r:** abre o arquivo apenas para leitura. Não permite escrita e, portanto, não atende ao requisito do estudante de ler e escrever no arquivo.
2. **rb**: abre o arquivo para leitura em formato binário. Semelhante ao modo r, mas não permite escrita, e por isso não é adequado para a necessidade do estudante.
3. **w+**: disponibiliza o arquivo para leitura e escrita, mas no processo de abertura, o arquivo é truncado (se existir), ou seja, seu conteúdo anterior é apagado. Isso não atende ao requisito de manter os dados existentes.
4. **rb+**: abre o arquivo para leitura e escrita em formato binário, mantendo o conteúdo existente. Isso permite que o estudante leia e escreva dados numéricos no arquivo sem perder as informações já armazenadas. Esse modo é ideal para manipular arquivos binários em que é necessário preservar o conteúdo ao abrir o arquivo.
5. **ab**: disponibiliza o arquivo para adição (append) em formato binário. Ele permite escrever no final do arquivo, mas não permite leitura nem modificar o conteúdo já existente, o que limita sua funcionalidade para a tarefa descrita.

Detalhamento do rb+:

- **Leitura e escrita**: combina tais capacidades, o que é essencial para a tarefa do estudante. Ele pode ler os dados existentes, modificar ou adicionar novos, e então, escrever essas modificações de volta ao arquivo.
- **Formato binário**: utilizá-lo assegura que as operações sejam feitas em formato binário. Isso é crucial quando se trabalha com dados numéricos e outros tipos que não são representados adequadamente em texto (como imagens ou dados estruturados).

- **Preservação de dados**: não apaga o conteúdo existente do arquivo ao abrir, o que é diferente de outros modos, como w+. Isso é fundamental para garantir que nenhum dado seja perdido durante o processo de manipulação.

A escolha do modo rb+ é justificada porque ele proporciona a funcionalidade completa e necessária para ler e escrever em um arquivo binário existente, sem comprometer os dados já armazenados, atendendo perfeitamente às necessidades do estudante para a tarefa em questão.

E/S por arquivos de texto

Em programação, manipular arquivos de texto é uma habilidade essencial, pois permite aos desenvolvedores ler, escrever, criar e modificar dados persistentes de maneira eficiente. Em C, a biblioteca stdio.h oferece várias funções robustas para essas operações. Por meio de exemplos de código, os programadores podem aprender a gerenciar os arquivos.

Para saber mais sobre o conceito de entrada e saída por arquivos de texto, Confira o vídeo a seguir.

Conteúdo interativo

Acesse a versão digital para assistir ao vídeo.

Abrir, criar e manipular arquivos em C

Para abrir ou criar um arquivo em C, utilizamos a função **fopen()**, que está definida no arquivo de cabeçalho stdio.h.

Na sequência, vamos explorar a sintaxe dessa função.

```
c

FILE *p = fopen("nome_do_arquivo", "modo_de_abertura");
```

Agora, acompanhe um exemplo prático!

```
c

FILE *p = fopen("Hello.txt", "r");
```

Também podemos criar um arquivo com **fopen(),** se ele não existir, especialmente quando usamos os modos w (escrita) ou a (anexação). Acompanhe!

```c
// Programa em C para criar um arquivo
#include
#include

int main() {
    FILE *ptr;
    ptr = fopen("./Hello.txt", "w");

    if (ptr == NULL) {
        printf("Erro ao criar o arquivo!");
        exit(1);
    }

    fclose(ptr);
    printf("Arquivo criado\n\n");

    return 0;
}
```

Para escrever em um arquivo, podemos usar **fprintf()** para formatar a escrita ou **fputs()** para inserir strings diretamente. Veja!

```c

// Programa em C para escrever em um arquivo
#include
#include

int main() {
    FILE *ptr;
    ptr = fopen("./Hello.txt", "w+");

    if (ptr == NULL) {
        printf("Erro ao escrever no arquivo!");
        exit(1);
    }

    char str[] = "Dados a serem inseridos no arquivo.";
    fputs(str, ptr);

    fclose(ptr);
    printf("Dados escritos no arquivo\n\n");

    return 0;
}
```

A função **fgets()** é frequentemente utilizada para ler linhas de texto de um arquivo. Ela lê até determinado limite de caracteres ou até encontrar uma quebra de linha (ou o que ocorrer primeiro).

```
c


// Programa em C para ler conteúdos de um arquivo
#include
#include

int main() {
    char str[80];
    FILE* ptr;

    ptr = fopen("Hello.txt", "r");

    if (ptr == NULL) {
        printf("Erro ao abrir o arquivo");
        // Se o ponteiro retornar NULL, o programa será encerrado
        exit(1);
    }

    if (fgets(str, 80, ptr) != NULL) {
        puts(str); // Exibe a string lida
    }

    fclose(ptr);

    return 0;
}
```

O programa tenta abrir o arquivo Hello.txt para leitura. Se o arquivo não puder ser aberto, ele exibirá uma mensagem de erro e encerrará o programa. Caso contrário, ele lerá até 79 caracteres ou até a primeira quebra de linha, imprimindo o que foi lido. O arquivo é, então, fechado ao final do processo, garantindo que todos os recursos sejam liberados corretamente.

Atividade 2

Em um curso de programação, uma aluna está aprendendo a manipular arquivos em C. Ela compreende a utilização da função fopen() para abrir ou criar arquivos e está familiarizada com os diferentes modos de abertura e suas especificidades. Durante um exercício prático, ela deve escolher o modo de abertura correto para executar uma operação específica no arquivo Hello.txt.

Se o objetivo da estudante é verificar se o arquivo Hello.txt existe e, em caso negativo, criá-lo sem escrever imediatamente nele, qual modo de abertura de arquivo ela deve usar com a função fopen(), a fim de minimizar o risco de perder dados existentes?

A

r

B

w

C

a

D

r+

E

w+

A alternativa C está correta.

O modo a (anexação) abre o arquivo para escrita no final do conteúdo existente, se ele existir, ou criar um novo se ele não existir. Esse modo não sobrescreve o conteúdo existente, minimizando o risco de perda de dados.

E/S por arquivos binários

Em linguagem C, a manipulação de arquivos binários desempenha importante papel em aplicações que exigem eficiência e precisão no armazenamento e na recuperação de dados. Diferentemente dos arquivos de texto — que armazenam informações de forma legível por humanos — os binários lidam com dados em seu formato bruto, permitindo operações mais rápidas e com menor consumo de espaço.

Para aprofundar seus estudos sobre o assunto, acompanhe, neste vídeo, o conceito de entrada e saída por arquivos binários.

Conteúdo interativo

Acesse a versão digital para assistir ao vídeo.

Gravar e ler dados de um arquivo binário em C

Vamos explorar como gravar e ler dados de um arquivo binário em C, utilizando as funções **fwrite()** e **fread()**, pois elas permitem trabalhar com dados em formato binário de maneira eficiente.

Para **gravar** em um arquivo binário, utilizamos a função fwrite(), que permite escrever blocos de dados de qualquer tipo diretamente no arquivo. A sintaxe ficará da seguinte maneira:

```
c

fwrite(const void *ptr, size_t tamanho_dos_elementos, size_t numero_de_elementos, FILE
*arquivo);
```

Acompanhe um exemplo de programa para gravação em C:

```c

// Programa C para gravar em um arquivo binário
#include
#include

struct Num {
    int n1, n2;
};

int main() {
    struct Num obj;
    FILE *fptr;

    if ((fptr = fopen("temp.bin", "wb")) == NULL) {
        printf("Erro ao abrir o arquivo");
        exit(1);
    }

    for (int n = 1; n < 10; n++) {
        obj.n1 = n;
        obj.n2 = 12 + n;
        fwrite(&obj, sizeof(struct Num), 1, fptr);
    }

    fclose(fptr);
    printf("Dados gravados no arquivo binário\n\n");

    return 0;
}
```

fread() é utilizada para ler dados de um arquivo binário. Confira, a seguir, a sintaxe dessa função:

```c

fread(void *ptr, size_t tamanho_dos_elementos, size_t numero_de_elementos, FILE
*arquivo);
```

Veja um exemplo de programa para leitura.

```
c


#include
#include

struct Num {
    int n1, n2;
};

int main() {
    struct Num obj;
    FILE *fptr;

    if ((fptr = fopen("temp.bin", "rb")) == NULL) {
        printf("Erro ao abrir o arquivo");
        exit(1);
    }

    for (int n = 1; n < 10; ++n) {
        fread(&obj, sizeof(struct Num), 1, fptr);
        printf("n1: %d\tn2: %d\n", obj.n1, obj.n2);
    }

    fclose(fptr);
    return 0;
}
```

Nos programas anteriores, manipulamos um arquivo binário chamado **temp.bin**.

No primeiro, gravamos dados no arquivo usando fwrite(), e no segundo, lemos os mesmos dados utilizando fread().

Isso garante que podemos recuperar exatamente os dados que foram gravados, demonstrando a eficácia das operações binárias em C para gerenciamento de dados estruturados.

Atividade 3

Em uma aula de programação avançada em C, um grupo de estudantes está explorando o uso de arquivos binários para armazenar e recuperar estruturas de dados complexas.

Ele aprendeu sobre as funções fwrite() e fread(), que são fundamentais para trabalhar com dados em formato binário. Durante um exercício prático, eles implementam um programa que grava e lê estruturas do tipo struct Num, que contém dois inteiros para um arquivo chamado temp.bin.

Ao revisarem o código que usa fwrite() e fread(), os estudantes precisam entender a importância de especificar corretamente os parâmetros dessas funções. Qual dos seguintes parâmetros é fundamental para garantir que os dados sejam gravados e lidos corretamente no arquivo binário?

A

O nome do arquivo, no qual os dados serão gravados ou lidos.

B

O ponteiro para a estrutura de dados, que será usada para leitura ou gravação.

C

O número de bytes de cada elemento a ser lido ou escrito.

D

O número total de elementos que deve ser lido ou escrito.

E

O modo de abertura do arquivo (wb para gravação, rb para leitura).

A alternativa C está correta.

Especificar o número de bytes de cada elemento a ser lido ou escrito é fundamental para garantir que os dados sejam manipulados corretamente no arquivo binário.

Vamos analisar o papel de cada parâmetro nas funções fwrite() e fread() para entender por que o número de bytes é tão importante.

1. **O nome do arquivo**: embora seja importante para abri-lo, o nome do arquivo não influencia diretamente como os dados são lidos ou gravados. Ele é especificado na função fopen(), e não em fwrite() ou fread().
2. **O ponteiro para a estrutura de dados**: é essencial para indicar onde os dados devem ser lidos ou gravados. No entanto, sem especificar corretamente o tamanho dos elementos (em bytes), fread() e fwrite() não saberão quantos bytes copiar de ou para essa estrutura.
3. **O número de bytes de cada elemento a ser lido ou escrito**: define o tamanho de cada elemento que está sendo lido ou escrito. Para garantir que a função manipule os dados corretamente, especialmente quando se trata de estruturas complexas, como struct Num, é essencial que o número de bytes corresponda ao tamanho da estrutura. Isso é geralmente obtido usando o operador sizeof(struct Num).
4. **O número total de elementos**: indica quantos elementos devem ser lidos ou escritos, mas sem o tamanho correto de cada elemento, a função não pode operar corretamente. Ele trabalha em conjunto com o tamanho dos elementos para processar os dados corretamente.
5. **O modo de abertura do arquivo**: é importante para definir a operação (leitura ou escrita) e o formato (binário ou texto). Contudo, ele é especificado na função fopen() nem diretamente em fwrite() ou fread(). Mesmo com o modo correto, os dados podem ser corrompidos se o tamanho dos elementos não for especificado corretamente.

Veja, agora, o detalhamento do papel do tamanho dos elementos:

- **Integridade dos dados**: especificar o número de bytes corretamente garante que cada elemento seja lido ou gravado integralmente. Isso é especialmente importante para estruturas de dados complexas, nas quais a falta de precisão pode resultar em dados corrompidos ou incompletos.

- **Coerência na operação entre leitura e escrita**: é mantida quando o tamanho dos elementos é corretamente especificado. Ao usar sizeof(struct Num), por exemplo, os estudantes asseguram que fread() e fwrite() manipulem a mesma quantidade de dados, preservando a integridade e a estrutura deles.

- **Facilidade de manutenção**: especificar corretamente o tamanho dos elementos facilita a manutenção e atualização do código. Se a estrutura mudar, ajustar o tamanho automaticamente atualiza a leitura e a escrita dos dados, minimizando o risco de erros.

Especificar o número de bytes de cada elemento a ser lido ou escrito é fundamental para garantir a integridade dos dados ao usar fread() e fwrite() em arquivos binários.

Demonstração prática de E/S por arquivos

Nesta prática, focaremos os fundamentos do sistema de arquivos e como a linguagem C pode ser utilizada para manipular tanto arquivos de texto como binários. Você poderá abrir, ler, escrever e fechar arquivos, utilizando essas habilidades em exemplos práticos que simulam aplicações reais — como a gestão de dados de uma empresa ou o armazenamento de informações de configuração de software.

Mas antes, assista ao vídeo e ao guia completo para as operações essenciais de entrada e saída por arquivos em C, abordando desde a criação até a manipulação avançada de dados. Veja também como as funções fopen(), fprintf(), fread() e fwrite() são utilizadas para explorar todas as nuances dessas operações fundamentais em programação.

Conteúdo interativo

Acesse a versão digital para assistir ao vídeo.

Roteiro de prática

Desenvolva um programa em C que atue como um sistema de registro para uma conferência. Porém, você deve levar em consideração algumas premissas:

- O programa deve ser capaz de salvar informações de participantes em um **arquivo de texto** e, simultaneamente, registrar o número deles em um **arquivo binário**.

- O programa também deve conseguir ler os arquivos para verificar o número atual de participantes e listar todos os registrados.

Chave de resposta

- O programa utilizará dois tipos de arquivos: um de texto para armazenar os nomes dos participantes e outro binário para manter o contador do número total de participantes.

- A abertura de ambos os arquivos está no código do programa, bem como a escrita dos nomes no arquivo de texto e a atualização do contador no arquivo binário. Além disso, caso o arquivo binário não exista, o programa será capaz de criá-lo e iniciar o contador.

- O programa exibirá, ao final, o total de participantes registrados, mostrando a integração eficiente entre leitura e escrita de arquivos de texto e binário, o que é essencial para aplicações que necessitam de manipulação de dados persistente e segura.

Agora, veja o código com a solução:

```c
#include


int main() {
    FILE *fpTexto, *fpBinario;
    char nome[100];
    int contador = 0;

    // Abertura dos arquivos
    fpTexto = fopen("participantes.txt", "a+");
    fpBinario = fopen("contador.bin", "rb+");
    if (fpBinario == NULL) { // Arquivo binário não existe, criá-lo
        fpBinario = fopen("contador.bin", "wb+");
       fwrite(&contador, sizeof(int), 1, fpBinario);
    } else { // Ler o contador existente
       fread(&contador, sizeof(int), 1, fpBinario);
    }

   printf("Digite o nome do participante: ");
    gets(nome); // Usando gets para simplificação, ideal usar fgets
    fprintf(fpTexto, "%s\n", nome); // Escreve o nome no arquivo de texto
    contador++;
    rewind(fpBinario);
   fwrite(&contador, sizeof(int), 1, fpBinario); // Atualiza o contador no arquivo binário

    // Fechamento dos arquivos
    fclose(fpTexto);
    fclose(fpBinario);

    printf("Total de participantes registrados: %d\n", contador);
    return 0;
}
```

Faça você mesmo!

Imagine que o programa de nossa demonstração prática esteja em uso. Qual modificação deveria ser realizada para a leitura do nome no programa sem prejudicar que o nome dos participantes possa incluir espaços?

A

Alterar gets(nome) para scanf("%[^\n]", nome);

B

Mudar de FILE * para fstream.

C

Usar puts(nome); em vez de gets(nome);

D

Implementar uma função personalizada para leitura de strings.

E

Nenhuma das alterações é necessária.

A alternativa A está correta.

O uso de scanf("%[^\n]", nome); permite a leitura de uma linha completa, incluindo espaços, até encontrar uma nova linha, o que resolve a limitação da função gets(nome). A função gets() lê uma linha inteira de entrada até encontrar um caractere de nova linha ('\n'). No entanto, gets() foi removida das versões mais recentes do C devido a problemas de segurança relacionados a estouro de buffer. Além disso, gets() não oferece proteção contra strings longas demais para o buffer.

Considerações finais

O que você aprendeu neste conteúdo?

1. Métodos de entrada (leitura) e saída (escrita) pelo console em C.
2. Domínio das funções básicas para E/S no console, reforçando a interação direta com o usuário.
3. Entendimento do padrão ANSI/ISO C e sua importância para a portabilidade do código.
4. Conhecimento sobre o padrão UNIX e sua influência no desenvolvimento de software.
5. Análise comparativa entre E/S ANSI/ISO C e UNIX C, identificando a melhor escolha conforme o contexto.
6. Exploração dos fundamentos do sistema de arquivos e sua estrutura em C.
7. Aprendizado sobre entrada e saída por arquivos de texto, utilizando as funções específicas da linguagem.
8. Capacidade de trabalhar com arquivos binários, abordando suas particularidades e vantagens.
9. Habilidade para escolher entre arquivos de texto e binários baseando-se nas necessidades do projeto.
10. Consolidação de técnicas práticas e teóricas para otimizar a utilização de E/S em projetos de C.

Explore +

Para se aprofundar ainda mais nesse assunto, leia o artigo **Padronização POSIX x sistemas operacionais de tempo real: uma análise comparativa**, que analisa aspectos do padrão POSIX.4 (1003.1b) e POSIX.4a (1003.1c), como comunicação entre processos, compartilhamento de memória, sincronização, escalonamento e threads, comparando-os com sistemas operacionais de tempo real comerciais e acadêmicos.

Referências

COPES, F. **The C beginner's handbook: learn C programming language basics in just a few hours**. freeCodeCamp, 9 mar. 2020. Consultado na internet em: 1 maio 2024.

DÖKME, Ç. **Enhancing the file access speed via direct I/O in Linux**. Medium, 18 fev. 2022. Consultado na internet em: 27 abr. 2024.

GEEKSFORGEEKS. **C - File I/O** . GeeksforGeeks, 20 mar. 2023. Consultado na internet em: 27 abr. 2024.

ISO - International Organization for Standardization. ISO/IEC 9899:1999. **Programming languages – C**, 1999.

ISO 9899 Wiki. **The Standard – C**. Consultado na internet em: 27 abr. 2024.

KUTTI, N. S. **C and Unix programming**: a comprehensive guide incorporating the ANSI and POSIX standards. *s. l.* : Cote Literary Group, 2002.

TRENTIN, P. M. **Aula 8 - E/S pelo Console**. Paulo Trentin, 8 mar. 2011. Consultado na internet em: 27 abr. 2024.

TWUMASI, A. **Exploring File I/O in C**. DEV, 12 out. 2023. Consultado na internet em: 27 abr. 2024.