Implementação e Otimização de Código em C para Cálculo de Imposto de Importação

Joao Eduardo Machado Dantas
Algoritmos e Estrutura de Dados Avançado
Anhanguera Campo de Marte

Resumo

Este artigo apresenta a implementação de um código em linguagem C para calcular o imposto de importação de produtos, bem como as otimizações realizadas para melhorar o desempenho e a eficiência do código. O código original foi analisado e refinado para reduzir o uso de memória e aumentar a velocidade de execução, mantendo sua funcionalidade e precisão.

Introdução

O cálculo do imposto de importação é uma tarefa comum em processos de importação e comércio internacional. Neste contexto, a linguagem de programação C oferece uma plataforma eficiente e flexível para implementar algoritmos que realizam esse cálculo de maneira precisa e rápida. Este artigo descreve a implementação de um código em C para calcular o imposto de importação de três produtos, bem como as otimizações aplicadas para melhorar sua eficiência.

Implementação Original

O código original implementa o cálculo do imposto de importação para três produtos distintos. Ele solicita ao usuário os valores dos produtos e a taxa de imposto em porcentagem, calcula o imposto para cada produto e exibe o resultado. Abaixo está o código original:

// **Código original**

```
#include <stdio.h>

float calcularImposto(float valorProduto, float taxaImposto) {
   return valorProduto * taxaImposto / 100.0;
}

int main() {
   float valorProduto, taxaImposto, impostoTotal;
   printf("Digite o valor do produto: ");
   scanf("%f", &valorProduto);
   printf("Digite a taxa de imposto em porcentagem: ");
   scanf("%f", &taxaImposto);
   impostoTotal = calcularImposto(valorProduto, taxaImposto);
   printf("O imposto de importacao a ser pago e: R$ %.2f\n", impostoTotal);
   return 0;
}
```

Descrição das Implementações Realizadas:

1. **Utilização de Estruturas de Dados:** Foram empregadas estruturas de dados como fila e pilha para armazenar os valores dos produtos e seus impostos correspondentes. Isso facilita o gerenciamento dos dados e permite o processamento sequencial dos valores.
2. **Separação dos Valores de Produto e Imposto:** Cada produto é inserido na fila, e o imposto correspondente é calculado e empilhado em uma pilha. Isso separa claramente os valores dos produtos dos impostos e simplifica o processo de cálculo.
3. **Reversão da Ordem dos Cálculos:** Após calcular os impostos para cada produto, os impostos são desempilhados e enfileirados novamente para restaurar a ordem original. Isso garante que os valores dos produtos e impostos correspondentes permaneçam alinhados corretamente.

Essas implementações proporcionam uma estrutura sólida para o cálculo do imposto de importação e servem como base para as otimizações subsequentes.

//**Código Implementado**

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

#define NUM_ITENS 3

// Estrutura para nó da fila
typedef struct Node {
   float valor;
   struct Node* next;
} Node;
// Estrutura para nó da pilha
typedef struct StackNode {
   float imposto;
   struct StackNode* next;
} StackNode;
// Função para enfileirar um valor
void enqueue(Node** rear, float valor) {
   Node* newNode = (Node*)malloc(sizeof(Node));
   newNode->valor = valor;
   newNode->next = NULL;
   if (*rear == NULL) {
      *rear = newNode;
   } else {
      (*rear)->next = newNode;
      *rear = newNode;
   }
}

// Função para desenfileirar um valor
float dequeue(Node** front) {
```

```
    if (*front == NULL) {
        printf("Fila vazia\n");
        return -1;
    }
    Node* temp = *front;
    float valor = temp->valor;
    *front = (*front)->next;
    free(temp);
    return valor;
}
// Função para empilhar um imposto
void push(StackNode** top, float imposto) {
    StackNode* newNode = (StackNode*)malloc(sizeof(StackNode));
    newNode->imposto = imposto;
    newNode->next = *top;
    *top = newNode;
}
// Função para desempilhar um imposto
float pop(StackNode** top) {
    if (*top == NULL) {
        printf("Pilha vazia\n");
        return -1;
    }
    StackNode* temp = *top;
    float imposto = temp->imposto;
    *top = (*top)->next;
    free(temp);
    return imposto;
}

// Função para calcular o imposto total
```

```
float calcularImpostoTotal(Node** front, float taxaImposto) {
    float impostoTotal = 0;
    StackNode* top = NULL;
    // Calcula o imposto de cada item e empilha na pilha
    for (int i = 0; i < NUM_ITENS; i++) {
        float valorProduto = dequeue(front);
        float imposto = valorProduto * taxaImposto / 100.0;
        impostoTotal += imposto;
        push(&top, imposto);
    }
    // Desempilha os impostos e enfileira novamente para restaurar a fila original
    while (top != NULL) {
        float imposto = pop(&top);
        enqueue(front, imposto);
    }
    return impostoTotal;
}
int main() {
    Node* front = NULL;
    Node* rear = NULL;
    float taxaImposto, impostoTotal;

    // Insere os valores dos produtos na fila
    for (int i = 0; i < NUM_ITENS; i++) {
        float valorProduto;
        printf("Digite o valor do %do produto: ", i + 1);
        scanf("%f", &valorProduto);
        enqueue(&rear, valorProduto);
        if (front == NULL) {
            front = rear;
        }
```

```
    }

    printf("Digite a taxa de imposto em porcentagem: ");
    scanf("%f", &taxaImposto);

    // Calcula o imposto total e exibe o resultado
    impostoTotal = calcularImpostoTotal(&front, taxaImposto);
    printf("\nO imposto de importacao total a ser pago e: R$ %.2f\n", impostoTotal);

    return 0;
}
```

Otimizações Realizadas

Para melhorar o desempenho e a eficiência do código, foram realizadas as seguintes otimizações:

1. **Redução do Uso de Memória:** Simplificou-se a estrutura de dados, eliminando a necessidade de armazenar os valores dos produtos em uma fila ou pilha. Os valores são calculados imediatamente após serem inseridos, reduzindo assim a quantidade de memória utilizada.
2. **Aprimoramento da Velocidade de Execução:** O cálculo do imposto é realizado em tempo real, enquanto os valores dos produtos são inseridos, eliminando a necessidade de percorrer a estrutura de dados duas vezes. Isso resulta em uma execução mais rápida do código.

Implementação Otimizada

O código otimizado apresenta uma estrutura mais simples e eficiente, mantendo a funcionalidade do código original. Abaixo está o código otimizado:

```
// Código otimizado
#include <stdio.h>

#define NUM_ITENS 3

int main() {
    float taxaImposto, impostoTotal = 0;
```

```
    printf("Digite a taxa de imposto em porcentagem: ");
    scanf("%f", &taxaImposto);

    for (int i = 0; i < NUM_ITENS; i++) {
        float valorProduto;
        printf("Digite o valor do %do produto: ", i + 1);
        scanf("%f", &valorProduto);

        float imposto = valorProduto * taxaImposto / 100.0;
        printf("Valor do %do produto (sem imposto): R$ %.2f\n", i + 1, valorProduto);
        printf("Imposto do %do produto: R$ %.2f\n", i + 1, imposto);
        impostoTotal += imposto;
    }

    printf("\nO imposto de importacao total a ser pago e: R$ %.2f\n", impostoTotal);

    return 0;
}
```

Conclusão

A implementação e otimização de códigos em C para calcular o imposto de importação demonstra a importância de considerar tanto a funcionalidade quanto o desempenho na criação de algoritmos. As otimizações realizadas foram bem-sucedidas em reduzir o uso de memória e aumentar a velocidade de execução do código, tornando-o mais eficiente e adequado para aplicações práticas.

Referências

1. Knuth, D. E. (1998). "The Art of Computer Programming, Volume 1: Fundamental Algorithms". Addison-Wesley.
2. Cormen, T. H., Leiserson, C. E., Rivest, R. L., & Stein, C. (2009). "Introduction to Algorithms". MIT Press.
3. Pereira, S. L. (2019). "Estrutura de Dados: Algoritmos, Análise da Complexidade e Implementações em C e C++". Editora Érica.