Universidade Católica de Brasília

Algoritmo e Programação Estruturada

Curso de Engenharia de Software

Trabalho N1

Decodificador de Mensagens

Autor: João Ricardo Jales Cirino, Felipe Lima Duarte
Orientador: Jerfferson Salomão Rodrigues

Brasília - DF
2024

JOÃO RICARDO JALES CIRINO

FELIPE LIMA DUARTE

DECODIFICADOR DE MENSAGENS

Trabalho apresentado ao curso de graduação em Engenharia de Software da Universidade Católica de Brasília, como requisito parcial para aprovação na matéria.

Brasília

2024

Referência: CIRINO, João Ricardo Jales; DUARTE, Felipe Lima. Decodificador de Mensagens, 2024. nr p. Bacharelado em Engenharia de Software – UCB – Universidade Católica de Brasília, Taguatinga – DF, 2024.

Resumo:

Projeto de decodificação de mensagens em hexadecimal utilizando a Linguagem C. Nesse estudo, foi utilizado conhecimentos de funções, limites de variáveis, *header files*, operações matemáticas, manipulação de variáveis e utilização de diferentes bibliotecas. O resultado do projeto foi a saída de uma mensagem decodificada.

1. ATIVIDADES

1.1 ANÁLISE DOS LIMITES SUPORTADOS PELOS TIPOS DE ARQUIVOS.

Para a conclusão da primeira atividade foi utilizado a biblioteca <limits.h> para que seja possível definir as constantes que delimitam os valores máximos e mínimos que variáveis de um dado tipo podem assumir. Dessa forma, podemos delimitar cada tipo de variável para uma função especifica sabendo a sua limitação superior e inferior. Além disso, foi analisado as 2 situações de "estouros" de variáveis, o **overflow** e o **underflow.**

Para a demonstração de valores maiores que o limite permitido foi declarado uma variável **x** do tipo **int** (inteiro), recebendo o seu limite máximo (INT_MAX) somado mais 1. Essa situação ultrapassa o limite permitido pela variável inteira ocorrendo o overflow. De acordo com Flausino e Mendes (2015), "Pode-se entender que o buffer overflow é o resultado do armazenamento de uma quantidade maior de dados que sua capacidade suporta.". Com a ocorrência do overflow, o sistema retorna o valor para o mínimo permitido (-2147483648). No caso de tipos **unsigend int** (inteiro positivo), ao somar 1 no seu valor máximo (4294967295), o valor retorna para 0. Tipos unsigned armazenam valores sem sinal, ou seja, valores positivos (MEIRA).

Para a demonstração de valores menores que o limite permitido foi declarado uma variável **y** do tipo **int,** recebendo o seu limite mínimo (INT_MIN) diminuído 1. Nesse cenário, ocorre o underflow onde a variável possui um valor menor do que o mínimo permitido. Com a ocorrência do underflow, o sistema retorna o valor para o máximo permitido (2147483647).

1.2 DESAFIO

1.2.1 Função que invalida alguns caracteres

Foi utilizado um *header file* Funcao.c e um cabeçalho Funcao.h para a implementação da função de invalidação de caracteres. Além disso, a biblioteca <math.h> foi usada nos cálculos exponenciais com o método pow( ) e para o arredondamento do retorno da função com o método round( ).

No arquivo Funcao.c foi declarado uma função funca_val( ) do tipo **int**, onde foi passado os 2 parâmetros previstos (int x, int b). A partir disso, foi declarado as variáveis **a0**, **a1**, **a2**, **a3** e **a4** do tipo **float** e as variáveis **a5**, **a6** e **a7** do tipo **double.** A

aplicação dos tipos float e double representam quantidades fracionarias (ponto flutuante) (SCHILDT, 2024). O tipo float utiliza geralmente 32 bits e sua precisão arredondada é de 7 dígitos decimais. Em tipos double sua precisão arredondada é de 15 dígitos decimais e geralmente é utilizado 64 bits. Além disso, foi declarado a variável **funcao** do tipo double para armazenar o valor gerado pelo cálculo da função.

1.2.2 Função principal

A função principal foi declarada no arquivo Decodificador.c e para a sua implementação foi usadas as bibliotecas <stdio.h>(entrada e saída), <stdlib.h> (método para conversão **strtol( )**) e, por fim, a declaração do cabeçalho "Funcao.h".

A partir disso, foi declarado dentro da função principal main( ), a variável **numMensagens** do tipo **Unsigned Int** (1, 10000) que indica a quantidade de mensagens interceptadas sendo sempre maiores que 1, as variáveis do tipo **int: x** e **b** (parâmetros da função func_val), **decimal** (Guarda o valor decimal dos caracteres em hexadecimal), **i** e **j** (Estruturas de repetição). Por fim, foi declarado as varáveis do tipo **char** (caracter) **mensagem[101]** (cadeia de caracteres {string}), onde é armazenada a mensagem completa com no máximo 100 caracteres em hexadecimal e **aux[3]** que é utilizada como auxiliar para a conversão, armazenando 2 caracteres temporários em hexadecimal.

A função decorre com a leitura (via scanf) de **numMensagem** (**%u** tipo unsigned), **b** (**%d** tipo inteiro) e **mensagem** (**%s** {string}). Após isso, ocorre a conversão da **mensagem**, a verificação pela função de invalidação func_val e, em seguida, a impressão da mensagem decodificada usando printf.

1.3 DESAFIO EXTRA

Na composição do desafio extra foi utilizado somente a biblioteca <stdio.h> que define a entrada e saída de dados. Foram declaradas as variáveis do tipo **float n1**, **n2**, **n3** e **ppd** que recebem as notas padrões do aluno. Para a nota do exame unificado foi declarado a variável **EU** recebendo o valor 0, devido a possibilidade de o aluno não ter realizado o exame e a variável **notaFinal** recebendo o valor final das notas. Por

fim, foi declarado as variáveis **verificadorEU** e **verificadorN3**, que auxiliam na validação da realização do exame unificado e na N3.

Para a validação requisitada em cada parte do desafio, foi utilizado a estrutura de repetição **do{ }while( )** junto à estrutura condicional **if ( )** , de modo que uma determinada condição fosse verificada e caso a mesma não seja validada é feito uma nova requisição de entrada. A formação da nota final foi feita a partir da somatória das notas N1 + N2 + PPD + EU. Caso o aluno tenha realizado a N3, a mesma estrutura de repetição e condição citada acima verifica todos os casos e determina se a nota de N3 em N1 ou N2 é substituída ou não. Também foi validado se **notaFinal** receber valores maiores que 10, limitando a nota final em 10. Por fim, caso sua nota for maior ou igual a 7 o aluno é considerado aprovado, caso contrário, reprovado. Assim, completando o desafio extra e implementando o sistema de avaliação da UCB.

2 CONCLUSÃO

Nesse projeto, foi possível conhecer as especificações dos tipos de variáveis e suas limitações, testando casos de **overflow** e **underflow**. O desafio principal e o desafio extra exigiram a aplicação de conhecimentos da linguagem C para imprimir a mensagem decodificada e implementar o sistema de avaliação da UCB, concluindo assim o trabalho.

REFERÊNCIAS

FLAUSINO, Helder Dias Costa; MENDES, Luís Augusto Mattos. Prevenindo e solucionando ataques de Buffer Overflow.

SCHILDT, Herbert. **C completo e total**. Makron, 1997.

MEIRA, Luıs Augusto Angelotti. Aula 03-Variáveis.

APENDICE

Repositório GitHub: https://github.com/JoaoJales/Decodificador-de-mensagens.git