Trabalho Prático I

Alunos: Felipe Melo & Raul Camizão

Disciplinas: Algoritmo e Estruturas de Dados & Introdução a Ciência da Computação

Curso: Ciência da Computação

Introdução

O homem contemporâneo está familiarizado com a base 10 (decimal), no dia-a-dia, já os computadores atuais trabalham exclusivamente com a base 2 (binário), assim é preciso fazer conversões entre estas bases quando se pretende inserir algum valor para ser processado pelo computador.

Em calculadoras e computadores, a conversão de números para o binário para a operação interna, e depois, para o decimal na visualização é um processo interno pré-programado para ser feito por ela. O ponto a ser entendido aqui é que internamente ela faz tudo em binário, em outras palavras: ela converte o que foi digitado para binário, faz o cálculo, converte o resultado para decimal e apresenta o resultado.

No entanto quando se está escrevendo um programa é normal a introdução de valores no meio do código, e em muitas situações a digitação de códigos binários é muito complicada/longa para o programador, então existem outros códigos que facilitam a digitação, na prática é muito utilizada a base 8 (octal), e a base 16 (hexadecimal), ambas derivadas da base 2 (note que estas bases facilitam a digitação somente, de qualquer forma ao ser compilado toda e qualquer base usada para escrever o programa é convertida para base 2 para que o valor seja usado pelo processador), apesar das mais usadas serem as bases 2, 8, 10, 16, existem *n* bases no sistema númerico.

Desta forma este Trabalho Prático teve como objetivo, implementar um Conversor de Bases em Linguagem C, que executa conversões de números reais entre as bases **decimal**, **binária**, **octal, hexadecimal** e uma **Base X** que pode ser criada pelo usuário da forma que o mesmo desejar.

Solução do Problema

Para a resolução do problema proposto pelo trabalho prático, foi implementado um algoritmo que primeiramente recebe uma cadeia de caractere (sempre contendo o sinal e o valor), o número correspondente à base de origem e o número correspondente à base de destino.

Feitas todas as verificações do conteúdo da *stdin* (existência do sinal, existência do ponto, verificação de inclusão do valor digitado ao conjunto de caracteres pertencentes à base de origem , condição de parada do programa e etc.) e retornando pela *stderr* a mensagem de erro, o programa entra em uma condição em que se a base de destino ou de origem for a Base X, o arquivo com os elementos desta base pré-definidos é aberto e lidos com o *fgetc()* e fechado após a leitura de todos elementos, neste procedimento o número da base e a posição/valor de cada elemento são guardados em variáveis.

Em seguida chama-se a função que transforma os caracteres no seu valor referente a ao seu valor no tipo inteiro. Isto é necessário pois cada valor do tipo char, possui um valor para representá-lo como um número inteiro.

Como pode ser visto neste fragmento da tabela ASCII:

ASCII Table

Dec	Char
48	0
49	1
50	2
51	3
52	4
53	5
54	6
55	7
56	8
57	9

Para um valor octal, como por exemplo: 246.0, se fossemos jogar algarismo por algarismo no somatório de conversão para base decimal que veremos a frente, teríamos 50 x 10^2 + 52 x 10^1 + 54 x 10^0. O que resultaria em um valor incoerente com o objetivo do trabalho. Desta forma é necessário sempre subtrair 48 do valor inteiro do char, para obtermos o valor correto do algarismo. Por outro lado, para o caso particular da base hexadecimal que utiliza os caracteres de 'A' a 'F', foi atribuído o valor específico correspondente ao mesmo.

Por meio de uma função que retorna ao programa o índice em que se encontra o '.' (ponto) que separa a parte inteira da fracionária, é possível separar a parte inteira da fracionária, atribuindo as ao tipo *int e float* respectivamente.

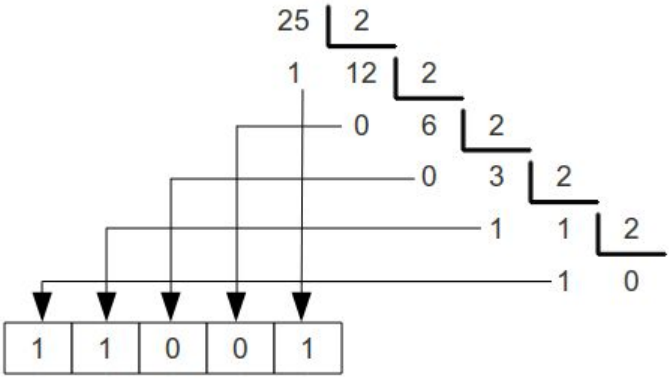
Valor:	+	1	0	**.**	1
Índice:	0	1	2	**3**	4

Independente da base de destino, é aplicado antes o algoritmo de conversão para a base decimal, primeiramente para a parte inteira e depois a parte fracionária, usando a posição do ponto como referência de término para a porção inteira e o fim da string como referência de término para a porção fracionária.

$$\sum_{i=1}^{n} a_i \times b^{i-1}$$

Obtido a conversão do conteúdo numérico da *string* para a base decimal, se a base de destino era a decimal, somamos a parte inteira com a parte fracionária e assim, temos o *loop* concluído.

Se não, devemos realizar mais uma conversão, primeiramente da parte inteira. Este método se baseia em **sucessivas divisões** do inteiro pela base de destino, guardando a cada divisão, o resto em um vetor, até que se obtenha zero da divisão do número inteiro pela base.
Como mostrado no exemplo prático a seguir:



Já para parte fracionária utilizamos o oposto da parte inteira, o método das **multiplicações sucessivas**. Esse método consiste em multiplicar a parte fracionária do número desejado pela base para a qual se deseja converter, até que a mesma seja ZERO.
O número convertido para a base desejada é igual a concatenação de todas as partes inteiras obtidas nos resultados das multiplicações, que também são guardadas no mesmo vetor, porém com o ponto separando-os da parte inteira obtida pelo método das divisões.

Por fim, antes de imprimir o resultado da conversão no arquivo de saída, precisamos transformar o vetor em um vetor de caracteres (*string*), para isso deve-se realizar o processo inverso do que foi feito no início do programa, somar 48 ao valor do *char*. Nos casos particulares da hexadecimal, para os *char* de 10 a 15 devemos atribuir de ‘A’ a ‘F’, isto é necessário para obtermos uma saída satisfatória. Se não fosse implementado este trecho de código ao programa, teríamos hipoteticamente o seguinte exemplo: se tivermos como entrada: +171.625 10 16, obteríamos na saída +1011.10, o que obviamente não queremos.

Já nos casos em que a base de destino é a base X, o vetor onde está os símbolos lidos do arquivo é acessado se o elemento do vetor for igual a uma das posições existentes na base X, o símbolo alocado na posição é imprimido.

Testes realizados

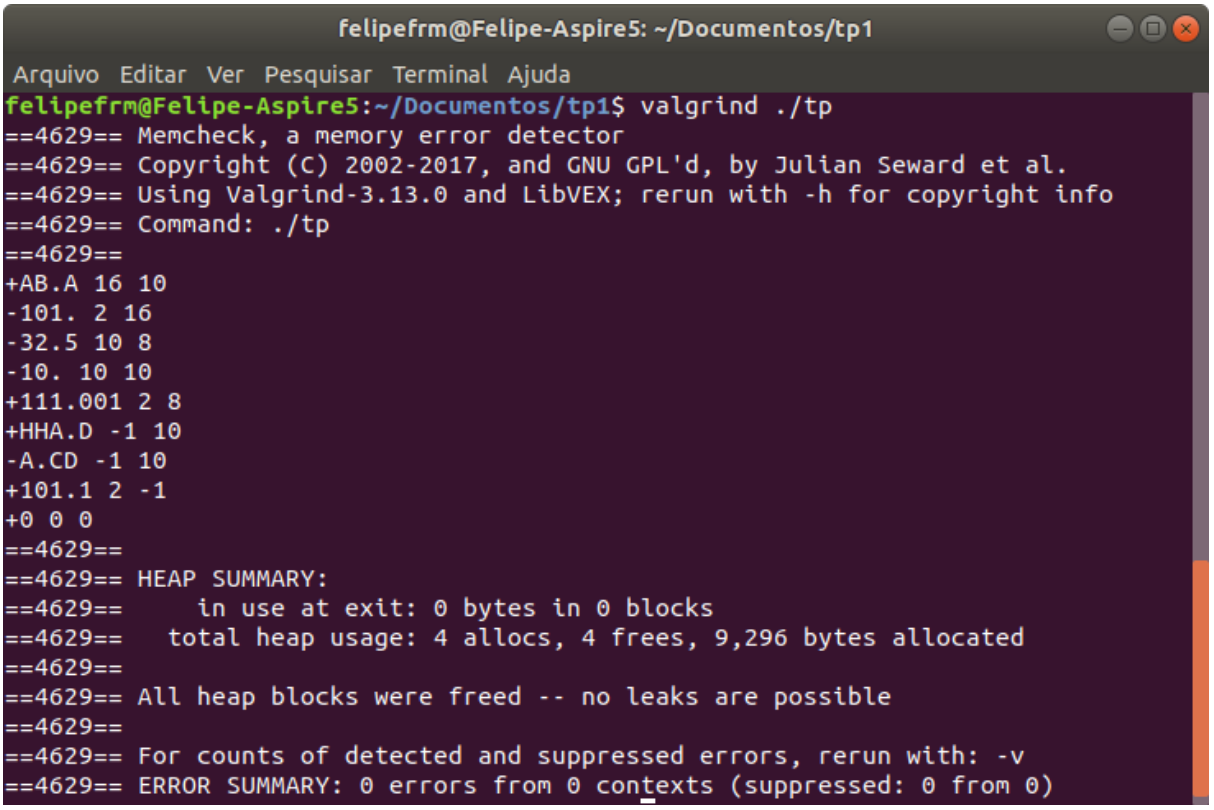
Os testes foram satisfatórios para os exemplos de entradas presentes na documentação de orientação do TP, e para várias outras entradas analisadas.

segue a imagem da execução do ./tp:

e abaixo o que foi impresso no arquivo de saída:

Os testes foram feitos entre todas as bases (bin-dec, bin-oct, bin-hex, oct-bin ... inclusive conversões envolvendo a base X), alternado entre números inteiros ou reais, sinal positivo ou negativo. Sempre conferindo o resultado da conversão com a “Calculadora” *BC* do terminal linux.
Foi testado também, casos em que o número (incluindo sinal, e ponto) é maior que 50 dígitos, o programa nesta situação não realiza a conversão e acusa uma mensagem de erro, implementada justamente para estes casos. Lembrando que as mensagem de erros do programa, saem pela *stderr*, a saída de erro padrão, que é independente à *stdout*, a saída padrão.

Análise *Valgrind*:



O relatório do *Valgrind* informa que 2 blocos de memória foram alocados e devidamente liberados. Sendo assim, não há vazamento de memória durante a execução do programa.

Referência Bibliográfica

As 10 conversões numéricas mais utilizadas na computação. Disponível em<https://dicasdeprogramacao.com.br/as-10-conversoes-numericas-mais-utilizadas-na-computacao/>. Acesso em: 25 maio 2018.

BACKES, André. **Linguagem C: completa e descomplicada**. Elsevier Brasil, 2012.