Trabalho Prático I
Manipulador de Conjuntos Numéricos

Gabriel Thiago H. Dos Santos, Gustavo H. Ferreira Cruz
<ra107774, ra109895>@uem.br

***Abstract*—Neste primeiro trabalho prático houve a implementação em linguagem *Gnu Assembly* para a plataforma 32 *bits* de um programa de manipulação de conjuntos de números inteiros, contendo a leitura de dois conjuntos e a partir disto realizando operações de União, Interseção, Diferença e Complementar com ambos conjuntos, além de printar os mesmos.**

***Index Terms*—Latex template, IEEE, Latin America Transactions**

I. Introdução

EM aplicações *Assembly* deve-se ter uma atenção maior, já que para a implementação o tratamento dos dados e sequencias podem influenciar bastante no desenvolvimento, como é o caso da pilha em que deve-se estruturar da maneira correta do devido programa. Para este trabalho a gerência é prioridade, já que os conjuntos A e B são definidos pelo usuário, com tamanho variados, com uso então de vetores dinâmicos e alocação com *Malloc*. E para cada opção oferecida aos usuários acaba por percorrer todos elementos dos conjuntos, exigindo a organização do desenvolvimento.

O seguinte relatório está organizado por sessões, no qual a II descreve o objetivo do trabalho, em III é abordado a descrição dos principais módulos desenvolvidos, em seguida em IV é mostrado os passos para execução. Em V realizamos uma autoavaliação do funcionamento da execução. Ao final, temos a conclusão.

II. Objetivos

O principal objetivo deste trabalho é praticar os conteúdos visto em sala de aula, principalmente com o uso de funções *call*/*ret*, uso de vetor dinâmico com *Malloc* e *Free* e *loops* em várias situações.

III. Descrição dos principais módulos desenvolvidos

O algoritmo se dá início com a chamada para impressão do menu de opções, seguida a leitura da opção escolhida e de uma chamada (*call*) para a função responsável por analisar a opção recebida pelo usuário e delegar funções para conceder a resposta para a opção solicitada pelo usuário. O módulo *_start* de início, responsável pelas chamadas das funções já comentadas termina com um comando de salto (*jmp*) para o início novamente, sendo que, só é possível sair do programa quando a opção certa é selecionada - neste caso, a opção 7 (sair do programa).

A função responsável por gerenciar as opções e chamar a função que corresponde a solicitação do usuário é composta por uma leitura da opção digitada pelo usuário anteriormente e a comparação encadeada com cada uma das funções das quais é possível chamar, isto é:

1) Leitura dos conjuntos A e B;
2) Descobrir a União;
3) Descobrir a Interseção;
4) Descobrir a Diferença de A - B;
5) Descobrir o complementar de B em relação a A;
6) Impressão dos Conjuntos Inseridos
7) Sair do Programa

A função de leitura dos conjuntos é uma chamada à função *_leConjuntos*, responsável por chamar a função de leitura duas vezes, uma para cada conjunto que pode ser inserido. Internamente, na leitura de cada conjunto, é realizado um *loop* com *scanf* para ler cada um dos elementos do conjunto, logo após definir o tamanho dele (primeiro é feito a leitura do tamanho e dos elementos conjunto A, e, depois, de B). Além disso, no meio da leitura de um conjunto (*_leConjunto*), é realizada a verificação de se é possível inserir o elemento fornecido pelo usuário - isso é, ele já não existe dentro do conjunto, vale ressaltar de que não são feitas verificações quanto ao tipo de dado ser ou não um inteiro válido. Para a verificação de elementos repetidos, chama-se a função *_insercaoUnica* que é responsável por capturar cada elemento digitado e comparar com cada um dos elementos já inseridos no conjunto (exceto quando é o primeiro elemento). Esse processo é feito utilizando um sistema de repetição com decremento nas posições do registrador de índice responsável por guardar o conjunto sendo formado. Vale salientar que, caso um elemento repetido seja inserido, a execução não é abortada, mas sim uma mensagem de erro é mostrada ao usuário e ele pode inserir um novo elemento sem consequências para o conjunto.

Para seguir as próximas opções é importante destacar de que deve-se ter preenchido os conjuntos ao menos uma vez, e isto é verificado e então não permitido seguir com as outras opções, exigindo o preenchimento dos conjuntos.

A função de Interseção se resume a uma iteração aninhada no conjunto A e B, ou seja, inicia o *loop* pelos elementos de A e para cada elemento é passado por *loop* a todos elementos do conjunto B, procurando pelos elementos repetidos em ambos. Uma vez encontrado um elemento que respeita a interseção, ele é *printado* como uma resposta e o laço de repetição continua.

A função de União utiliza a ideia parecida com a função de interseção para achar a resposta, ou seja primeiro é feito uma busca procurando cada um dos elementos de A que não

estão em B, depois, o conjunto B é *printado* por inteiro, completando a União.

A função de Diferença executa em *loop* percorrendo para todo elemento do conjunto A e então para cada elemento de B, comparando-os, se o elemento de de A estiver em B é setado uma variável auxiliar *temEmB* para um, e então depois que percorrer todo conjunto B é verificado se esta variável está com o valor um, se estiver equivale que há elemento repetido, caso o valor seja zero o elemento questão de A é *printado* na tela, e então após isso é setado a varíavel padrão para zero e o *loop*, tendo ao final o conjunto resposta da diferença de A - B.

A função do cálculo do Complementar é realizada considerando o A como o conjunto que abrange o B, ou seja, B deve estar contido em A. Primeiro é verificado se o conjunto B é maior que o conjunto A - se verdadeiro, finaliza-se a execução da função e o usuário é redirecionado para o menu principal, pois não é possível calcular o complementar de um conjunto que não está contido no outro. Com a etapa dos tamanhos concluída, é feito uma repetição aninhada para verificar se o conjunto B possui algum elemento que não está contido no conjunto A. Se encontrado um elemento que está em B mas não existe em A, a execução da função é cancelada, uma mensagem de erro é impressa ao usuário e, por fim, ele é redirecionado para o menu de opções. Para a repetição aninhada, usou-se a interação no conjunto B para a repetição externa e a iteração no conjunto como a repetição interna, comparando um a um os elementos de B para com os elementos de A. Uma vez que ambos os conjuntos atendam aos requisitos do complementar, é feita o cálculo dele - isto é, busca-se todos os elementos de A que não estão em B.

A função de Impressão é simplesmente uma repetição que imprime um a um cada elemento em um conjunto, replicada para ambos os conjuntos.

A última função é a que implementa o código de Saída, simplesmente redirecionando o fluxo para o final do algoritmo, que chama a função verifica se houve definição de conjuntos e caso sim, é desalocado os vetores com *Free* (a mes a função é utilizada na opção 1 em momentos de digitar novos conjuntos) e depois encerra a execução com o código de sucesso (0).

IV. Execução

Para a execução do arquivo:

1) Vá a raiz do projeto;
   Para a compilação em 32bit do programa:
2) as -32 source.s -o file
   Agora, para a *linkagem* do programa:
3) ld -m elf_i386 **file** -l c -dynamic-linker /lib/ld-linux.so.2 -o **executavel**
   Se tem então o executável, execute com:
4) ./executavel

Com isso então iniciará o programa e pode-se ir nas opções desejadas.

V. Auto-avaliação do funcionamento

Como auto-avaliação, podemos considerar o trabalho implementado como um algoritmo que tem sucesso em atender as especificações e aplicar os conceitos visto em sala de aula. Destaca-se como pontos positivos o uso da memória de maneira prudente, não deixando resquícios ou inconsistências de memória e a aplicação dos conceitos ensinados em sala de aula, como desalocação de memória para os vetores com o *free*, e como também o uso de *call* com *ret* quando possível, evitando quebra de fluxos e vícios de programação que fazem o código ficar confuso e mais difícil de compreender (sentidos na prática da implementação - não há necessidade de deixar uma linguagem que já é complexa, ainda mais complexa) ou empilhamento e desempilhamento de dados para proteger os dados contidos dentro, uma vez que funções como o *printf*, da biblioteca C, tendem a alterar estes valores. Como pontos a se melhorar, podemos citar a falta de opções para o usuário gerenciar as operações sobre os conjuntos, isto é, não existem opções que façam, por exemplo, a diferença ser feita de B para A ou vice-versa (o mesmo vale para o complemento) e os trechos de códigos parecidos que, com ajustes, poderiam se tornar única função únicas mais genéricas, economizando algumas linhas de código.

Conclusão

Finalizamos o trabalho seguindo completamente a recomendação com todas funcionalidades exigidas funcionando de maneira correta e com isso a prática foi atingida da melhor maneira, conhecendo melhor a linguagem e seu modo de processar as instruções e como se deve gerenciar para atingir os objetivos.