ESCOLA POLITÉCNICA DA USP

SISTEMAS DE PROGRAMAÇÃO

PCS3216

SEGUNDA PROVA

GRUPO 6 - TEMA I - EXPANSOR DE MACROS

DAVID HENRIQUE DA COSTA - 10774170

LUCAS CANOSSA CIPOLLA - 10769542

PEDRO HENRIQUE FLORIO MENDES - 10770990

São Paulo, 2 de Agosto de 2021

Sumário

1. Apresentação e Fundamentos

Enunciado do exercício:

Especificar uma notação para a definição de macros simples com parâmetros, e construir um expansor que identifique a definição das macros e efetue sua expansão, produzindo um texto isento de referências a essas macros.

a. Contextualização

Na área da programação, pode-se notar que é comum haver repetição de certas lógicas utilizadas dentro de um mesmo programa. Muitas vezes encontramos estruturas muito similares que aparecem repetidas vezes dentro de um mesmo código, a exemplo, uma estrutura que realiza as chamadas para imprimir algo na tela do computador, para ler dígitos do usuário ou a passagem de parâmetros para realizar a chamada de uma subrotina. Repetir essas estruturas diversas vezes num mesmo código, além de ser trabalhoso para o programador, torna o código de mais difícil visualização e entendimento; caso alguém precisasse fazer uma alteração no código do programa teria um grande trabalho para compreender o código e até mesmo encontrar a parte onde deve realizar a alteração.

É nesse contexto que surgem as macros nos sistemas de programação. As macros são nada mais do que uma definição de um termo que, pelo resto do programa, será substituído por um conjunto de instruções correspondentes. No exemplo citado anteriormente, uma macro que serve para imprimir algo na tela do computador, por exemplo, seria uma macro de grande utilidade. Com o uso de macros, aumentamos um pouco o nível de abstração do programa, mas diminuímos o trabalho do programador e deixamos o código mais limpo e de fácil compreensão.

Podemos citar ainda, outra grande vantagem das macros. Caso por alguma razão, o programador tenha errado algo no trecho de código que se repete ao longo do programa. Se ele transformá-o numa macro, corrigir a macro será o suficiente para corrigir o funcionamento de todas as vezes que ela é chamada, evitando que o programador precise passar por todo o código

corrigindo a sequência de instruções a cada vez que ela aparece, tarefa que além de trabalhosa é provável de resultar em erros.

b. Definições e Bases Teóricas

Tipos de Macros conforme a aplicação:

1. **Macros Léxicas**

Utilizamos as macros léxicas para simplificar nosso texto e aumentar o nível de abstração. O modelo se fundamenta em associar nomes a padrões textuais, o qual substitui a chamada por um expansão da mesma. Os argumentos também são tratados como padrões, porém temporários, gerado à medida que a macro é expandida e extinta após procedimento.

2. **Macros sintáticas**

São utilizadas em linguagens de programação com sintaxe extensível. Não serão tratadas neste projeto em específico.

Tipos de Macros Léxicas

1. **Macros Simples**

Macros simples são macros que não possuem parâmetros, ou seja, elas apenas realizam o processo de substituir uma palavra por um texto específico dentro de um determinado documento.

2. **Macros Paramétricas**

Neste caso, a macro se associa a uma lista externa de parâmetros, tratados como chamadas de macros. Assim, a macro é expandida de acordo com os argumentos.

3. **Macros Dependentes de Macros**

Neste tipo de macro, é possível que internamente à própria macro ocorra a chamada de outra ou outras macros. Isso eleva um pouco o nível de complexidade no tratamento da macro, já que possibilita macros aninhadas, e portanto, o programa deve conseguir determinar em que nível de aninhamento se encontra no momento

4. **Macros Recursivas**

Macros recursivas são aquelas que possibilitam a chamada da própria macro a ser definida dentro dela mesmo. Novamente isso eleva um pouco o nível de complexidade do programa que irá tratar das macros.

Outras definições

1. **Declaração de macros**

Denomina-se declaração de macros um trecho de texto que faz a associação da palavra-chave, ou nome de uma macro ao texto a qual ela corresponde, ou seja, faz associação direta entre a palavra que define a macro e o trecho de texto por qual essa palavra deve ser substituída. Vale ressaltar que é necessário uma sintaxe específica para a declaração de macros a fim de que o programa consiga reconhecer corretamente.

2. **Expansor de macros**

Denomina-se expansor de macros o programa que cumpre a função de, dadas determinadas declarações de macros, realizar a substituição de fato das palavras-chave de cada macro pelo texto correspondente, seja ele dependente ou não de parâmetros.

c. O funcionamento de expansores de macros

Neste trecho explicaremos mais tecnicamente como funcionam expansores de macros, para que seja mais simples a compreensão do expansor de macros projetado neste trabalho. O conteúdo abordado neste tópico é teórico e nem todo ele será aplicado no projeto, como será melhor explicado posteriormente.

O primeiro passo de um programa que servirá como expansor de macros é analisar o texto da declaração de macros, e então guardar na memória as palavras-chave que correspondem a macros, os textos das macros e os argumentos de cada macro. Dessa maneira, não será necessário analisar novamente o texto de declaração de macros, dado que o conteúdo dele já estará armazenado na memória do computador. Vale

ressaltar que para esse passo ser concluído corretamente, deve-se ter definido o modelo das declarações de macros.

Para isso, de maneira geral, divide-se o texto das declarações, e percorre ele, em procura de um padrão que represente a definição de uma macro. Encontrando isso, guarda-se o nome dela. Em geral, então, encontram-se os parâmetros da macro, seguindo algum padrão pré estabelecido. Após guardar o nome e os parâmetros da macro, continua-se lendo o texto e guardando-o na memória como texto associado àquela dada macro, até que se encontre um padrão que representa o final da macro.

O segundo passo será então realizar de fato a expansão das macros, ou seja, percorrer um texto que contém referências a macros que agora já estão guardadas na memória, e substituí-las pelo texto correspondente, também armazenado na memória do computador. Vale ressaltar que, novamente, é exigido um padrão, para que dessa maneira, o expansor consiga identificar os argumentos de uma determinada macro.

Entrando em mais detalhes do segundo passo, ele em geral ocorre da seguinte maneira: primeiro divide-se o texto em palavras distintas. Em seguida, para cada uma das palavras, procura-se uma referência a ela nos nomes de macros guardados na memória. daí existem duas opções: caso a palavra não corresponda a um nome de macro, copia-se ela para o texto; caso ela corresponda, entra-se no processo de expandir a macro, ou seja, é copiado no texto de saída o texto correspondente àquele nome de macro.

Caso a macro seja paramétrica, mais um passo é necessário, muito similar ao passo anterior: divide-se o texto da macro em si em palavras distintas, e em cada uma delas realiza-se o seguinte: caso a palavra não esteja na lista de argumentos, copia-se ela normalmente para o texto, caso ela esteja nessa lista, substituísse a palavra pelo argumento passado na macro naquele texto

Há ainda mais complexidade envolvida quando falamos de macros que podem referenciar outras macros, e especialmente macros recursivas. Nesses dois casos, ao percorrer o texto relacionado a uma macro, além de buscar por referências em suas palavras a argumentos dela, deve-se procurar também referências a nomes de macros. É importante também ter

nesses casos variáveis de controle que indiquem o nível de aninhamento ou de recursão no código a cada momento.

Quando tratamos especificamente de macros recursivas ou com referências cíclicas, há mais cuidados que devem ser tomados. Na declaração da macro, deve existir alguma comparação que sirva como condição de saída da macro. Ou seja, alguma condição faz com que a macro não faça outra referência a ela mesma novamente. Isso para evitar que uma macro recursiva apenas entre em um loop infinito e escreva um texto sem fim, fazendo referência a ela mesma a cada vez que chamada.

d. Objetivos

Temos por objetivo neste trabalho construir um programa expansor de macros simples e de macros paramétricas. Ou seja, devemos inicialmente criar uma definição de macros padronizada, e então criar um programa que primeiramente lê declarações de macros, e cria associações entre as palavras-chave das macros, o corpo de seu texto, e os possíveis parâmetros da macro; e em seguida, lê um texto que contém essas palavras chave e realiza a substituição destas pelo respectivo trecho de texto correspondente à macro.

Nota-se que, para a realização do programa, não se assume nada quanto a linguagem utilizada no programa que contém as macros, ou seja, na realidade a entrada do programa pode muito bem ser um documento que não contém um código, e sim um trecho de texto ou qualquer coisa parecida. O expansor, portanto, não deve fazer nenhuma consideração quanto ao formato do texto, apenas realizar o trabalho de trocar as palavras correspondentes às macros pelos textos que elas representam, com os respectivos parâmetros. O programa deve poder ser utilizado, portanto, também, como um recurso de “localizar/substituir” em um texto qualquer, dadas as devidas declarações de macros.

2. Programa Desenvolvido

a. Funcionamento do Programa

Para o expansor, tomamos primeiramente um arquivo contendo as declarações das macros a serem utilizadas. As macros do arquivo serão então incluídas em uma lista de macros e, caso ela tenha parâmetros, estes serão extraídos e registrados na lista de parâmetros das macros. Caso não tenha, será registrado como uma string nula na lista de parâmetros.

Com o programa já alimentado com as macros, carrega-se o arquivo de texto no qual o expansor de macros será utilizado. Em seguida, o programa percorre as palavras do arquivo, testando se as mesmas configuram uma chamada para as macros apresentadas na lista de macros armazenadas. Caso negativo, escreve a palavra normalmente no arquivo de saída. Já caso positivo, testa se a macro possui ou não parâmetros.

Para a condição de uma macro simples, busca na memória do programa qual deve ser a substituição a ser realizada para a macro em questão, e a escreve no arquivo de saída. Para as macros paramétricas, busca quais são os parâmetros da macro em questão e como eles devem ser escritos no arquivo de saída, e então as substitui pelo seu respectivo valor, passado na chamada da macro.

Resumindo o funcionamento do programa:

- ➔ Carrega as macros de um arquivo .txt;
- ➔ Carrega o arquivo de texto no qual será executado;
- ➔ Testa palavra a palavra por uma chamada de macro;
- ➔ Se não for, escreve-a no arquivo de saída;
- ➔ Se for, busca por argumentos:
    - ◆ Macros simples: substitui o texto e escreve no arquivo de saída;
    - ◆ Macros paramétricas: procura os parâmetros da macro, e os substitui pelos passados na chamada, e escreve no arquivo de saída

b. Padronização das macros

Para conseguirmos trabalhar com o texto inserido, é necessário adotar algumas padronizações quanto à declaração e chamada de macros. Quanto a declaração, adotaremos a seguinte padronização:

macro soma(a1,a2):
a1 + a2
endmacro

Neste caso, nosso programa identifica que é uma macro devido ao nome “macro”, identifica o nome como expressão entre “macro” e a abertura de parênteses, no caso, “soma”. Além disso, o programa reconhece a abertura de parênteses como parâmetros, neste caso a1 e a2, e armazena-os na lista de parâmetros. A partir do “:”, o mesmo reconhece que é o corpo da macro. Assim, nosso programa lê linha após linha e guarda o texto interno para utilizar numa próxima chamada de macro. O programa sabe que finalizou quando encontrar uma linha com “endmacro”, a qual não deve ser transcrita à chamada.

Para as chamadas de macros utilizaremos:

soma(20,30)

Assim, o programa identifica que deve expandir a macro quando encontra uma palavra seguida de um parêntese que está presente na lista de macros, que foi previamente criada, e dessa maneira pode substituir sua chamada pelo texto correspondente à macro em questão, com os devidos parâmetros. Para macros que não possuem argumentos, o parêntese deve ser aberto e fechado sem nenhum conteúdo dentro:

macro()

Caso haja algum erro no número de argumentos de uma determinada macro ou no modo de declaração, o programa tentará rodar normalmente, mas imprimirá na saída um aviso de erro no lugar do texto normal da macro.

c. Abstração do problema

Abaixo, vemos um pseudocódigo feito sobre o programa elaborado:

```
função ler macros(nome do arquivo):

  abre o arquivo com as declarações de macros

  carrega as linhas do arquivo

  listaMacros = []
  listaArgMacros = []
  listaTextMacros = []

  para cada linha:

    caso as 5 primeiras letras da linha = "macro":
      adiciona o nome da macro para a listaMacros

      a partir do nome até o fim da linha:
        caso o parâmetro exista:
          adiciona à lista de parâmetros da macro
          específica

    entre ':' e 'endmacro': #corpo da macro
      adicionar linhas da macro à lista listaTextMacros

  fecha o arquivo

  retorna as listas de macros, parâmetros e textos das
macros
```

```
função susbstituição(nome arquivo de entrada, nome do arquivo de saida, listaMacros, listaArgMacros, listaTextMacros):

  abre o arquivo com o texto

  cria o arquivo de saída

  carrega as linhas do arquivo de entrada

  para cada linha:

    divide em palavras
    para cada palavra:

      caso tenha um '(' após a palavra:
        verifica se a palavra está na lista de macros:
          carrega os parâmetros passados na chamada desta
          macro

      se palavra não está na lista de macros:
          escreve a palavra no arquivo de saída

      senão:
        percorre a lista de macros armazenada:
          encontra uma correspondência para a palavra
          atual:
            separa os argumentos passados na chamada:

            verifica se a quantidade de argumentos da
            chamada e da declaração é compatível:
              se não for, escreve uma mensagem de erro
```

```
                caso esteja correto:
                  carrega o corpo da macro:
                    separa em palavras

                    para cada palavra no corpo da macro:

                      verifica se a palavra é ou não um
                      parâmetro:
                        se nao for, escreve no arquivo de
                        saída

                      se for:
                        percorre a lista de argumentos da
                        macro:

                          verifica qual o argumento referido:
                            escreve o argumento passado na
                            chamada no arquivo de saída
  fecha os arquivos

main:

  chama a função de ler macros

  chama a função de substituição
```

O código foi dividido em duas funções principais: uma para ler e carregar as macros declaradas (ler macros no pseudocódigo), e outra para tomar o texto e utilizar a funcionalidade do expansor de macros (definida como substituição no pseudocódigo).

A primeira recebe um arquivo .txt com declarações de macros devidamente padronizadas, como mostrado acima, e as carrega em listas, separando em:

- Lista de Macros: contém os nomes das macros, a serem utilizados para a chamada das mesmas;
- Lista de Argumentos ou Parâmetros: contém os parâmetros necessários de cada macro, para as macros paramétricas. No caso das macros simples, contém uma string nula no lugar;
- Lista de Textos das Macros: contém o corpo das macros, sendo estes ou não parâmetros. Será utilizada para expandir a macro no arquivo de saída, quando a mesma é chamada no arquivo de entrada.

Já a segunda função recebe as listas da função anterior e percorre todo o texto passado em um arquivo de entrada .txt palavra a palavra, comparando-as com a Lista de Macros, para encontrar chamadas de macros no meio do texto. Ao encontrar, realiza os trabalhos de verificação da ocorrência de parâmetros, detecção de erros de sintaxe e quantidade de argumentos destoantes (podendo estes estarem na declaração ou na chamada da macro, caso não sigam o padrão definido), e, enfim, faz a expansão da macro com base nas listas recebidas, tanto para macros simples, quanto para macros paramétricas.

d. Dinâmica de operação

Com exemplos, iremos demonstrar alguns usos e forma de funcionamento do programa:

Exemplo 1:

Numa cidade chamada Inconfidentes, no sul de Minas Gerais, há um campus do Instituto Federal. O nome oficial a ser utilizado pela instituição é "*Instituto Federal de Educação, Ciência e Tecnologia do Sul de Minas Gerais - Campus Inconfidentes*". Suponhamos que, ao redigir um documento, seja necessário escrever este nome completo sempre que for necessário referenciar ao campus. Para isso, o expansor de macros pode ser utilizado.

Primeiramente declaramos a macro no arquivo destinado a isso da seguinte forma:

macro ifinc():
Instituto Federal de Educação, Ciência e Tecnologia do Sul de Minas Gerais - Campus Inconfidentes
endmacro

Agora, ao redigir um texto e rodar o expansor de macros, não será mais necessário digitar todo o nome da instituição. Basta digitar ifinc() onde o nome deve ser inserido, assim, ao encontrar a chamada, o expansor de macros realizará a substituição:

Entrada: Aulas presenciais do ifinc() voltarão em 2022.

Saída: Aulas presenciais do Instituto Federal de Educação, Ciência e Tecnologia do Sul de Minas Gerais - Campus Inconfidentes voltarão em 2022.

Exemplo 2:

Seguindo no ramo educacional, podemos também utilizar macros para equações de segundo grau, por exemplo.

Sabendo que as equações são da forma *"a * x ^ 2 + b * x + c = 0"*, podemos apenas criar uma macro paramétrica para escrever a equação completa, passando apenas *a*, *b* e *c*.

Declaramos então:

macro eq2(a,b,c):
*a * x ^ 2 + b * x + c = 0*
endmacro

Sendo assim, em uma lista de exercícios, por exemplo, obtemos o seguinte:

Entrada: Exercícios

1) Resolva as seguintes equações:
   a) eq2(4,8,11)
   b) eq2(6,1,1)

c) eq2(34,56,99)

Saída: Exercícios

1) Resolva as seguintes equações:

a) 4 * x ^ 2 + 8 * x + 11 = 0

b) 6 * x ^ 2 + 1 * x + 1 = 0

c) 34 * x ^ 2 + 56 * x + 99 = 0

3. Testes e Análise dos Resultados

Para realização dos testes, utilizamos uma função comum no contexto de macros: editoração. Nosso texto se baseia em aplicações didáticas, onde o editor simplifica seu processo de escrita, auxiliando em dois fatores principais: explicação de teorias e expressões matemáticas.

Oferecemos duas possibilidades pro escritor: a primeira são funções que não necessitam de argumentos, onde apenas colocando a abreviação do conteúdo, é escrito o nome por extenso e um link é gerado para a página onde há um desenvolvimento melhor sobre o tema que é utilizado nesta página. Esta aplicação é muito útil quando considera-se que as matérias são cumulativas, e o aluno depende de um conceito anterior para entender o fundamento que procura, assim, geramos impacto no aprendizado.

A segunda opção é baseada em LaTeX: colocando uma abreviação da expressão matemática com os argumentos certos, podemos gerar expressões complexas sem grande necessidade de aprofundamento do escritor. Assim, o editor não precisa conhecer especificamente o desenvolvimento da fórmula, apenas sabendo onde os argumentos se encaixam, ele pode gerar uma expressão completa, auxiliando em fórmulas mais complexas. Não só isso, num contexto de bibliotecas, também pode-se comparar nosso exemplo com a biblioteca math utilizada na programação. Com ela, o próprio programa consegue calcular expressões complexas com apenas os argumentos. Inserindo apenas mais uma variável de retorno, também conseguimos gerar algumas funções desta biblioteca.

Sendo assim, segue os três textos:

macros.txt:

```
macro soma(a1,a2):
a1 + a2
endmacro

macro eq2(a,b,c,d):
a x² + b x + c = d
endmacro

macro prodnot(a1,a2):
( a1 + a2 )²
endmacro

macro juntaBase(b1,b2,ba):
( b1 + b2 )/( ba )
endmacro

macro mcq():
Método de Completar Quadradados (para entender melhor:
https://brasilescola.uol.com.br/matematica/metodo-complet
ar-quadrados.htm)
endmacro

macro tqp():
Trinômio Quadrado Perfeito (para entender melhor:
https://brasilescola.uol.com.br/matematica/trinomio-quadr
ado-perfeito.htm)
endmacro
```

```
macro pn():
Produto Notável (para entender melhor:
https://brasilescola.uol.com.br/matematica/produtos-notaveis.htm)
endmacro
```

Aqui, realizamos a declaração de 7 funções:

soma(a1,a2): insere a soma de duas expressões;
eq2(a,b,c,d): insere uma equação de segundo grau por extenso;

prodnot(a1,a2): insere o produto notável de duas variáveis;

juntabase(b1,b2,ba): insere a soma de b1 e b2 divididos por uma mesma base;

mcq(): insere um link para a teoria do Método de Completar Quadrados;

tqp(): insere um link para a teoria dos Trinômios Quadrados Perfeitos;

pn(): insere um link para a teoria dos Produtos Notáveis

Segue, também, texto base:

```
Demonstração da Fórmula de Bhaskara

Partiremos inicialmente sobre uma equação de segundo grau
eq2(a,b,c,0) genérica.

Primeiramente, para utilizarmos o mcq() devemos dividir
nossa equação genérica por a, fazendo a passagem:

eq2(a,b,c,0) -> eq2(a/a,b/a,c/a,0/a) -> eq2(1,b/a,c/a,0)
```

Após isso, para conseguirmos o tqp(), dividimos b/a por 2 e elevamos ao quadrado, resultando em:

eq2(1,b/a,c/a,0) -> eq2(1,b/a,b²/4a²,-(c/a)+b²/4a²)

Assim, escreveremos a primeira parte da equação como um pn():

prodnot(x,b/2a)

e a segunda parte da equação colocaremos na mesma base:

soma(-c/a,b²/4a²) = soma(-c*4a/a*4a,b²/4a²) = juntaBase(-4ac,b²,4a²)

Dessa maneira, após fazer o cálculo da raiz quadrada podemos simplificar:

√ prodnot(x,b/2a) = √ juntaBase(-4ac,b²,4a²)

soma(x,b/2a) = ±(√(soma(-4ac,b²)))/(2a)

Portanto, isolando x de um lado da equação, chegamos facilmente em:

x = (-b ± √(-4ac+b²))/2a

Nesta parte, conseguimos ver como o editor escreveria o texto. Percebe-se que fica mais fácil para apresentar os conteúdos, visto que é preciso apenas saber a abreviação da teoria, também auxilia na busca pelo link da teoria, que é gerado automaticamente.

Também vê-se que as expressões matemáticas ficam muito mais intuitivas e fáceis de escrever, reduzindo o trabalho do escritor. Pode-se assemelhar muito nossa aplicação com um editor de páginas, como o HTML,

onde os textos são redigidos manualmente, porém a linguagem deixa a página mais intuitiva e facilmente escrita.

Assim, podemos prosseguir para nossa saída:

```
Demonstração da Fórmula de Bhaskara

Partiremos inicialmente sobre uma equação de segundo grau
a x² + b x + c = 0 genérica.

Primeiramente, para utilizarmos o Método de Completar
Quadradados (para entender melhor:
https://brasilescola.uol.com.br/matematica/metodo-complet
ar-quadrados.htm) devemos dividir nossa equação genérica
por a, fazendo a passagem:

a x² + b x + c = 0 -> a/a x² + b/a x + c/a = 0/a -> 1 x²
+ b/a x + c/a = 0

Após isso, para conseguirmos o Trinômio Quadrado Perfeito
(para entender melhor:
https://brasilescola.uol.com.br/matematica/trinomio-quadr
ado-perfeito.htm) dividimos b/a por 2 e elevamos ao
quadrado, resultando em:

1 x² + b/a x + c/a = 0 -> 1 x² + b/a x + b²/4a² =
-(c/a)+b²/4a²

Assim, escreveremos a primeira parte da equação como um
Produto Notável (para entender melhor:
https://brasilescola.uol.com.br/matematica/produtos-notav
eis.htm)

( x + b/2a )²
```

e a segunda parte da equação colocaremos na mesma base:

$-c/a + b^2/4a^2 = -c*4a/a*4a + b^2/4a^2 = (-4ac + b^2)/(4a^2)$

Dessa maneira, após fazer o cálculo da raiz quadrada podemos simplificar:

$\sqrt{}(x + b/2a)^2 = \sqrt{}(-4ac + b^2)/(4a^2)$

$x + b/2a = \pm(\sqrt{}(-4ac + b^2))/(2a)$

Portanto, isolando x de um lado da equação, chegamos facilmente em:

$x = -(b \pm \sqrt{}(-4ac+b^2))/2a$

Finalmente, podemos ver como o texto chega ao nosso usuário. Caso ele queira entender melhor algum conceito utilizado neste desenvolvimento, o link está a sua disposição. Além disso, as passagens matemáticas estão melhor desenvolvidas, passo a passo, para conseguir entender a demonstração da fórmula.

Finalmente, geramos uma plataforma educacional que otimiza o trabalho de escrita do editor, e o entendimento do usuário, apenas utilizando o conceito básico de macros. Entrando no mérito das macros recursivas, onde poderíamos até usar as próprias macros como argumentos de outras, nosso processo seria muito melhor simplificado. Também podemos utilizar imagens ou elementos visuais para facilitar o entendimento do aluno, que podem ser incluídas neste conceito.

Quanto aos resultados individualmente, notamos que nosso expansor se baseia em um conceito muito básico na sua função paramétrica: substitui toda palavra igual o argumento base pelo argumento novo. Isso pode gerar

alguns problemas quanto à declaração de variáveis, a qual precisamos que o compilador entenda onde precisa substituir cada variável.

Assim, é necessário que todas as variáveis estejam separadas do resto do texto, pois caso contrário o programa pode ter uma interpretação errônea. Porém, quanto ao contexto que estamos limitados (apenas um processador de macros léxicas simples e paramétricas, com mera substituição do texto), nosso programa funciona perfeitamente, e serve para os mais diversos propósitos que um expansor de macros possa abranger.

4. Conclusões

Nesta prova, construímos um expansor de macros simples e paramétricas não atrelado a nenhuma linguagem de programação específica, podendo assim ser utilizado em textos comuns, se fazendo extremamente útil para textos com demasiadas repetições ou expressões iguais dependendo de parâmetros variáveis.

O modelo construído associa nomes à padrões textuais, substituindo a chamada por um expansão da mesma.

A metodologia de trabalho aqui empregada dividida em três partes principais:

- ➔ Abstrações e padronizações:

  Nesta etapa, discutimos o que deveria ser feito, e como seria implementado. Além disso, definimos um padrão de como as macros seriam criadas e executadas, com a finalidade de evitar erros de sintaxe ao trabalhar com o expansor de macros.

- ➔ Construção do programa:

  Como modelo definido na etapa anterior, partimos para a criação do código para construir o nosso expansor de macros. A linguagem escolhida foi Python, por ser bastante simples e intuitivo, além de possuir ótimas ferramentas para tratamento de strings e manipulação de arquivos. Por se tratar de um projeto em conjunto, utilizamos também a extensão “Live Share” da IDE “Visual Studio Code”,

desenvolvida pela Microsoft para facilitar trabalhos colaborativos à distância.

➔ Depuração e testes:
Com o programa pronto, foi-se necessário a realização de testes com diferentes situações, partindo das mais simples para as mais incomuns, com o objetivo de efetuar a correção de possíveis erros, fazendo assim com que o expansor de macros funcione como desejado.

Por fim, obtivemos um programa estável e completamente funcional, dado o escopo do projeto. Como vimos no item 3 deste relatório, encontramos resultados altamente satisfatórios, poupando um grande trabalho ao demonstrar a Fórmula de Bháskara. Colocando em números, o texto de entrada possui 900 caracteres, enquanto o texto final após a expansão das macros tem 1269. Uma diferença de 369 caracteres, que equivale a 41% do texto original.

Levando em consideração o tamanho relativamente pequeno do texto de exemplo, podemos elevar essa porcentagem ainda mais para textos maiores nos quais se apliquem a funcionalidade do expansor de macros, fazendo com que nosso objetivo inicial tenha sido executado com sucesso.

5. Substitutiva

Como solicitado pelo professor, o aluno Lucas Canossa Cipolla (10769542) incrementou o código com conteúdo do tema J: Derivador de sentenças. Assim, aqui será feito um breve resumo sobre este tema com base no modelo comum do relatório:

Enunciado

• Decomponha regras gramaticais múltiplas em várias regras simples:

<expressão> ::= <expressão> + <termo>

| <expressão> - <termo>

| <termo>

pelas uma das três regras abaixo:

<expressão> ::= <expressão> + <termo>

<expressão> ::= <expressão> - <termo>

<expressão> ::= <termo>

• Use a gramática obtida para derivar um texto da linguagem definida pela gramática, partindo do seu não-terminal raiz, e repetidamente aplicando o roteiro abaixo até que não mais restem não-terminais no texto.

– O usuário escolhe para ser expandido um dos não-terminais presentes no texto

– O expansor apresenta ao usuário as possíveis regras de substituição aplicáveis

– O usuário escolhe uma dessas opções

– O expansor, tratando o não terminal como chamada de macro, e usando como corpo dessa macro o lado direito da regra escolhida, expande o não terminal efetuando a substituição escolhida

Contextualização

Da mesma maneira que o expansor de macros léxicos construído anteriormente, as aplicações se baseiam na simplificação de textos. Neste caso, ao invés do editor ter que configurar o código diretamente, para ele são disponibilizadas opções para substituir seu código, dependendo da aplicação que ele deseja no momento.

Esta aplicação de macros com escolhas pode auxiliar desde escolha de expressões matemáticas, até na própria editoração previamente mencionada, onde o usuário pode escolher as opções para a escrita.

Fundamentos

Para os fundamentos, muito se repetem os mesmos do expansor de macros. No nosso caso, temos macros simples ou com parâmetros, que podem ser escolhidos pelo usuário. Assim, os conceitos teóricos se mantém a não ser pelas diferenças de linguagem: neste caso, utilizamos uma notação mais robusta: com os comparadores “maior que” e “menor que” entre as expressões, para um melhor entendimento do compilador.

Assim, nosso programa funciona com as mesmas bases de expansão, porém dentro de uma mesma linha, ou seja, ao invés de substituir um texto no lugar da expressão, mantemos a expressão que chama, e apenas mudamos o lado direito.

Nosso objetivo é garantir a liberdade ao usuário quando utiliza uma função, não se limitando à mesma lógica, por exemplo do nosso exemplo das expressões matemáticas. Assim, caso quiséssemos unir a função soma() com a função subtração(), seria possível dentro do derivador de sentenças.

Com esta utilidade, também estamos cada vez mais próximos de um compilador real, mesmo que ainda faça falta uma variável de retorno das expressões que utilizamos. Porém, mesmo assim, acreditamos que há grande facilidade ao editor para editoração, simplificando uma mesma padronização de chamada para diferentes expressões.

O Programa Desenvolvido

Portanto, para conseguirmos entender com mais facilidade o nosso código, criamos um módulo a parte (que pode ser facilmente integrado com o código do expansor léxico simples) com as partes que tratam a expressão de maneira geral.

Assim, o programa:

- Lê um arquivo txt, linha a linha
- Identifica onde há expressões
- Armazena as opções de cada expressão
- Solicita ao usuário resolver cada ocorrência
- Substitui no arquivo a escolha do usuário

Nota-se que a lógica é até relativamente mais simples que o expansor léxico, devido a uma lógica padronizada para o derivador de sentenças: anteriormente, precisávamos tratar macros simples e marcos paramétrica.

Assim, a base do nosso pseudocódigo:

```
função ler macros(nome do arquivo):

  abre o arquivo com as declarações de sentenças

  carrega as linhas do arquivo

  para cada linha:

    caso a linha comece com = "<":
      adiciona o nome da expressão para a listaExp

      a partir do nome até o fim da linha:
        adiciona a primeira opção ao usuário

    entre '|' e o final da macro:
      armazena linha a linha como opções para o usuário

  fecha o arquivo

  retorna as listas, parâmetros e textos das expressões

função escolhe(nome da expressao, lista de expressões,
lista das opções, lista das quantidades das expressões)

identifica qual o nome na lista de expressões para
conseguir sua ordem
```

```
solicita ao usuário uma escolha das expressões
disponíveis

retorna o nome da expressão com a escolha do usuário

função susbstituição(nome arquivo de entrada, nome do
arquivo de saida, listaExp, listaArgExp, listaQuantExp):

  abre o arquivo com o texto

  cria o arquivo de saída

  carrega as linhas do arquivo de entrada

  para cada linha:

    divide em palavras
    para cada palavra:

      caso tenha um '<' no começo da linha:
        verifica se a palavra está na lista de macros:
          carrega os parâmetros passados na chamada desta
          macro

        percorre a lista de macros armazenada:
          encontra uma correspondência para a palavra
          atual:

          verifica se já entrou com o nome dessa
          expressão

            verifica qual a escolha do usuário:
            escreve a escolha no arquivo
```

```
  fecha os arquivos

main:

  chama a função de ler exp

  chama a função de substituição
```

Portanto, nota-se muita similaridade com o expansor léxico anteriormente quanto ao pseudocódigo, apenas substituindo a parte que seleciona os argumentos para a escolha do usuário de expressões.

Um exemplo simples para ser dado é quanto à tradução de palavras em um texto, tendo a palavra “Bom dia” pode-se escolher:

Bom Dia :: = Bom Dia

| Good Morning

| Ohayo

| Buenos Dias

Neste caso, sabendo a língua que se deseja traduzir o texto (supondo que este pode ser disponibilizado em várias linguagens), pode-se facilmente redigir um texto simples, tendo as palavras e suas respectivas traduções.

Testes e Análise dos Resultados

Portanto, seguem os textos simples de teste do programa com aplicação para a tradução:

exemplo.txt:

```
<Bom-Dia> ::= <Bom-Dia>
| <Good-Morning>
| <Ohayo>

<Boa-Noite> ::= <Boa> + <Noite>
| <Good> + <Night>
```

```
| <Buenas> + <Noches>
```

saida.txt (dadas as escolhas 2 e 2):

```
<Bom-Dia> ::= <Good-Morning>

<Boa-Noite> ::= <Good> + <Night>
```

O programa também foi testado com o exemplo dado pelo professor no slide, sendo:

<expressão> ::= <expressão> + <termo>
| <expressão> - <termo>
| <termo>

Também foram feitos testes com:

<termo> ::= <termo> + <expressão>
| <termo> - <expressão>
| <expressão>

Tivemos resultados dentro do esperado com todos estes testes.

Conclusão

Portanto, dentro das expectativas, tivemos um avanço significativo na nossa aplicação. Quando pensamos na possibilidade de juntar o expansor léxico com o derivador de sentenças, vemos que se tornam ainda mais expressivos os resultados.

Apesar de a parte incrementada ter sido sucinta, devido a diversas similaridades com o programa original dentro dos conceitos teóricos, tentamos inovar dentro do contexto com aplicações diferentes, como a tradução de palavras.

Assim, com o código funcionando, e a fácil integração com o expansor léxico, considera-se o derivador de sentenças um avanço muito importante dentro do seu contexto de aplicação.