Relatório da Tarefa 1A

Alunos: Gustavo de Souza; Felipe Jojima.

Questão 1:

O desenvolvimento foi feito em ambiente Linux utilizando a plataforma Vscode, porém também efetuamos testes utilizando a seguinte linha de comando:

gcc -Wall pilha.c arq.c -Larq.h && ./a.out

Importante salientar que tivemos problemas nos nossos testes e descobrimos que é preciso fazer uma modificação na inclusão no arquivo pilha.c dependendo do ambiente do teste como é mostrado abaixo:

```
1
2 #include "arq.h"
3 //se for compilar no terminal do linux deve se usar #include "arq.h"
4 //se for compilar vs code deve se usar #include "arq.c"
5 //caso contrario ele retornará problema de mais de uma chamada para as funcoes
6
```

Utilizamos para o desenvolvimento 3 arquivos : “pilha.c”, “arq.h” e “arq.c”.

A principal diferença no nosso arq.h para o que se encontra no moodle como modelo a ser seguido é como estruturamos nossa struct MP:

```
typedef struct mp
{
    int topo1;
    int tamInfo;
    int topo2;
    noPi *vet;
    int *vetAux;
    DescPilha pilha1, pilha2;

} MP;
```

Aqui criamos um vetor do tipo noPi que será nosso vetor principal onde terá armazenado dados do tipo Int e/ou Char se o usuário desejar (ambas as opções podem ser selecionadas no nosso menu) e um vetor chamado vetAux do tipo inteiro onde será nosso auxiliar que armazenará 0 caso o valor que o usuário deseja inserir seja um inteiro ou 1 caso seja um char, facilitando assim nossa organização em momentos de exibição do vetor.

Partindo para a análise do arq.c iremos detalhar com comentários cada função:

Função para contar o número de elementos do vetor:

```c
int numElementosInseridos(MP *desc)
{
    //teste de verificacao de existencia da multipilha
    if (desc == NULL)
    {
        printf("Multipilha inexistente!\n\n");
        return -1;
    }

    int qnt = 0;
    //conta a quantidade de elementos da pilha 1
    if (desc->topo1 > -1)
    {
        qnt += desc->topo1 + 1;
    }

    //conta a quantidade de elementos da pilha 2
    if (desc->topo2 < desc->tamInfo)
    {
        qnt += (desc->tamInfo - desc->topo2);
    }

    //retorna a soma dos elementos
    return qnt;
}
```

A função cria() tem algumas verificações que a deixam extensa então vamos dar atenção para a parte importante:

```c
                else//é possivel criar
                {

                    desc->topo1 = -1; //topo da pilha 1
                    desc->topo2 = tamInfo; //topo da pilha 2 comeca no final do vetor
                    desc->tamInfo = tamInfo;
                    aux = desc->vet;
                    //topo das subpilhas
                    desc->pilha1.topo = -1;
                    desc->pilha2.topo = -1;
                    for (int i = 0; i < tamInfo; i++)
                    {
                        (aux + i)->dados.inteiro = 0; //inicializa os inteiros do vetor aux como nulo
                        (aux + i)->dados.letra = '\0'; //inicializa os inteiros do vetor aux como "nulo"
                        desc->vetAux[i] = -1;
                    }
                    //retorna o ponteiro para multipilha recem criada
                    return desc;
                }
            }
        }
    }

    return NULL;
}
```

No trecho acima inicializamos as variáveis necessárias corretamente e utilizamos o nosso vetor aux que nos auxilia a descobrir se o valor inserido pelo usuário será um inteiro ou um char.

A função insereNaPilha() também possui suas verificações iniciais entao vamos dar atenção pra parte crucial que foi o nosso raciocínio para incremento do topo:

```
if (escPilha == 1)//se o usuario escolheu a pilha 1
{
    //incrementa o topo
    p->pilha1.topo += 1;
    p->topo1 += 1;

    p->pilha1.pontTopo = &(p->vet[p->topo1]);//atualiza o topo da
    pilha 1

    if (p->pilha1.topo == 0)//ve se é o primeiro da pilha 1
    {
        p->pilha1.pontTopo->ant = NULL;//nao tem anterior
    }
    else//atualiza o novo e o antigo topo
    {
        p->pilha1.pontTopo->ant = &(p->vet[p->topo1 - 1]);
        p->pilha1.pontTopo->ant->prox = p->pilha1.pontTopo;
    }

    p->pilha1.pontTopo->prox = NULL;//proximo nó é null

    //copia o novo elemento para o topo
    p->pilha1.pontTopo->dados = (*novo);
    p->vet[p->topo1].dados = (*novo);
    return 1;
}
```

E aqui caso o usuário desejar inserir na pilha 2:

```
else if (escPilha == 2)//se foi escolhido a pilha 2
{
    //incremente o top
    p->pilha2.topo += 1;
    p->topo2 -= 1;

    p->pilha2.pontTopo = &(p->vet[p->topo2]);//ponteiro do topo
    apontara para o novo

    if (p->pilha2.topo == 0)
    {
        p->pilha2.pontTopo->ant = NULL;
    }
    else//atualiza os ponteiros do topo novo e do anterior
    {
        p->pilha2.pontTopo->ant = &(p->vet[p->topo2 + 1]);
        p->pilha2.pontTopo->ant->prox = p->pilha2.pontTopo;
    }
    p->pilha2.pontTopo->prox = NULL;
    //copia do novo para o topo da pilha
    p->pilha2.pontTopo->dados = (*novo);
    p->vet[p->topo2].dados = (*novo);
    return 1;
}
```

A função removeDaPilha() foi desenvolvida usando o seguinte raciocinio para a pilha 1:

```
if (nPilha == 1)
{
    if (p->pilha1.pontTopo->ant != NULL)//verifica se a pilha tem mais de um elemento
    {
        p->pilha1.pontTopo = p->pilha1.pontTopo->ant;//atualiza o topo para apontar para o anterior
        //copiamos os dados removidos do no para a variavel "removido"
        memcpy(&(*removido), &(p->pilha1.pontTopo->prox->dados), sizeof(info));
        //atualiza a pilha ao anular os ponteiros do no removido
        p->pilha1.pontTopo->prox->ant = NULL;
        p->pilha1.pontTopo->prox = NULL;
    }
    else//so tem um elemento
    {
        //copiamos os dados removidos do no para a variavel "removido"
        memcpy(&(*removido), &(p->pilha1.pontTopo->dados), sizeof(info));

        //limpa os ponteiros do topo
        p->pilha1.pontTopo->ant = NULL;
        p->pilha1.pontTopo->prox = NULL;
    }
    //anula os dados do vet principal da posicao anterior
    p->vet[p->topo1].dados.inteiro = 0;
    p->vet[p->topo1].dados.letra = '\0';
    //decrementa os indices do topo da pilha 1
    p->pilha1.topo -= 1;
    p->topo1 -= 1;
}
```

E esse para a pilha 2:

```
else//escolheu a pilha 2
{
    if (p->pilha2.pontTopo->ant != NULL)//verifica se tem mais de um elemento
    {
        //atualiza para apontar para o no anterior
        p->pilha2.pontTopo = p->pilha2.pontTopo->ant;

        //copia os dados do no removido
        memcpy(&(*removido), &(p->pilha2.pontTopo->prox->dados), sizeof(info));
        //desvenciula o no removido
        p->pilha2.pontTopo->prox->ant = NULL;
        p->pilha2.pontTopo->prox = NULL;
    }
    else
    {
        //copia os dados do no removido
        memcpy(&(*removido), &(p->pilha2.pontTopo->dados), sizeof(info));
        //atualiza o topo
        p->pilha2.pontTopo->ant = NULL;
        p->pilha2.pontTopo->prox = NULL;
    }

    //limpa o vetor principal
    p->vet[p->topo2].dados.inteiro = 0;
    p->vet[p->topo2].dados.letra = '\0';
    //decrementa o topo da pilha 2
    p->pilha2.topo -= 1;
    p->topo2 += 1;
}
```

A função consulta topo é simples e rápida:

```c
int consultaTopo(MP *p, int pilhaAlvo, info *consultado)
{
    //verifica a multipilha
    if (p == NULL)
    {
        printf("Multipilha inexistente!\n\n");
        return 0;
    }

    if (pilhaAlvo == 1)
    {
        if (p->topo1 == -1)//não ha topo
        {
            printf("A pilha esta vazia!\n\n");
            return 0;
        }
        //topo sera definido aqui
        (*consultado) = p->vet[p->topo1].dados;
    }
    else
    {
        if (p->topo2 == p->tamInfo)//topo da pilha vazio
        {
            printf("A pilha esta vazia!\n\n");
            return 0;
        }
        //topo definido aqui
        (*consultado) = p->vet[p->topo2].dados;
    }
    return 1;
}
```

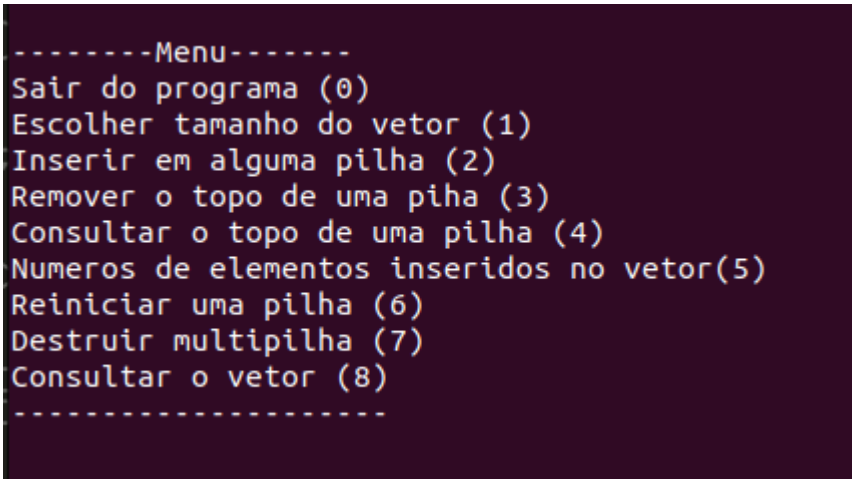
A função reinicializacaoPilha() que possui o seguinte corpo principal:

```c
if (escPilha == 1)
{
    // reinicializando a pilha 1
    for (int i = 0; i <= a->topo1; i++)
    {
        //zera os elementos do vetor auxiliar e do principal
        a->vetAux[i] = -1;
        a->vet[i].dados.inteiro = 0;
        a->vet[i].dados.letra = '\0';
    }
    //redefinicao dos ponteiros do topo
    a->pilha1.topo = -1;
    a->topo1 = -1;
    a->pilha1.pontTopo = NULL;
}
else
{
    //reinicializando a pila 2
    for (int i = a->topo2; i < a->tamInfo; i++)
    {
        //zera os elementos do vetor auxiliar e do principal
        a->vetAux[i] = -1;
        a->vet[i].dados.inteiro = 0;
        a->vet[i].dados.letra = '\0';
    }
    //redefinicao dos ponteiros do topo
    a->pilha2.topo = -1;
    a->topo2 = a->tamInfo;
    a->pilha2.pontTopo = NULL;
}
```

E por fim a função destroi():

```c
MP *destroi(MP *p)
{
    free(p->vet);//libera o vetor principal
    free(p->vetAux);//libera o auxiliar
    return NULL;//retorna null para dizer que a estrutura foi destruida
}
```

Partindo agora para uma analise completa do arquivo pilha.c vamos iniciar com o nosso menu e comentar sobre nossas implementações. Segue em anexo o menu em execução:



O usuario deve selecionar um valor entre 0 e 8 condizente com a ação, caso for digitado 0 o programa será encerrado.

Caso o valor for 1, teremos as seguintes linhas de codigo para a manipulação do tamanho do vetor principal:

```c
case 1://selecionar o tamanho do vetor principal
    int tam = 0;
    if (criado == 1)
    {
        printf("O vetor ja foi criado!!!\n");
        break;
    }
    printf("Insira o tamanho do vetor que deseja criar:\n");
    scanf("%d", &tam);
    mps = cria(tam);

    if(mps == NULL){
        printf("Não foi possivel criar o vetor!!!\n");
    }else{
        printf("Vetor criado com sucesso!\n");
        criado = 1;
    }

    break;
```

Detalhe importante, só é possível fazer isso uma vez no código, a menos que o usuário destrua a multi pilha com a opção 7 e escolha definir o tamanho novamente.

Caso o número inserido seja 2 entraremos na área de inserção em alguma pilha onde teremos os seguintes passos:
-Seleção de qual pilha inserir
-Seleção de inteiro ou char para inserir
-Escaneamento da entrada do usuário

E quando completo os passos anteriores iremos ter o seguinte trecho de código onde &novo é o valor de entrada do usuário:

```
    }
    int verify = insereNaPilha(mps, 1, &novo);
    if (verify == 0)
    {
        printf("FALHA AO INSERIR NA PILHA!\n");
        break;
    }
    printf("SUCESSO AO INSERIR NA PILHA!\n");

    if (escT == 0)
    {
        mps->vetAux[mps->topo1] = 0;
    }
    else
    {
        mps->vetAux[mps->topo1] = 1;
    }

    break;
```

Caso o valor for 3, além das verificações padrões teremos o seguinte segmento principal desse caso a seguir para efetuar uma remoção de um topo de uma pilha:

```
if (escPilha == 1)//pilha 1 escolhida
{
    int indiceT = mps->topo1;
    if (removeDaPilha(mps, 1, &removido) == 0)//erro
    {
        printf("Erro na remocao!!\n\n");
        break;
    }

    printf("Removido com sucesso!\n");
    if (mps->vetAux[indiceT] == 0)
    {
        printf("Item removido: %d\n", removido.inteiro);
    }
    if (mps->vetAux[indiceT] == 1)
    {
        printf("Item removido: %c\n", removido.letra);
    }
    mps->vetAux[indiceT] = -1;
}
else//pilha 2 escolhida
```

Caso o valor for 4, além das verificações teremos a chamada da função consultaTopo() e a nosso desenvolvimento foi:

```
if (consultaTopo(mps, escPilha, &consultado) == 0)
{
    printf("Topo inexistente!\n");
    break;
}

if (escPilha == 1)//consulta pilha 1
{
    printf("Consultando o topo da pilha 1:\n");
    if (mps->vetAux[mps->topo1] == 0)
    {
        printf("Elemento: %d\n\n", consultado.inteiro);
    }
    else
    {
        printf("Elemento: %c\n\n", consultado.letra);
    }
}
else//consulta pilha 2
{
    printf("Consultando o topo da pilha 2:\n");
    if (mps->vetAux[mps->topo2] == 0)
    {
        printf("Elemento: %d\n\n", consultado.inteiro);
    }
    else
    {
        printf("Elemento: %c\n\n", consultado.letra);
    }
}
break;
```

Caso o valor for 5, teremos a contagem do número de elementos do vetor principal, dado por:

```
case 5:
    int a = numElementosInseridos(mps);//retornara o numero de elementos
    if (a < 0)
    {
        printf("\nFalha na contagem!!\n\n");
        break;
    }
    printf("\nNumero de elementos inseridos no vetor: %d\n\n", a);
    break;
```

Caso o valor for 6 será efetuado a reinicialização de uma pilha desde que o usuário insira uma pilha que exista para que assim possa ser reiniciada:

```
case 6:
    escPilha = -1;
    printf("\nQual pilha deseja reinicializar (1 ou 2)?\n");
    scanf("%d", &escPilha);
    while (escPilha != 1 && escPilha != 2)//verificacao da entrada
    {
        printf("Escolha um numero valido! (Insira 1 ou 2)\n");
        scanf("%d", &escPilha);
    }
    if (reinicializacaoPilha(mps, escPilha) == 0)
    {
        printf("FRACASSO NA REINICIALIZAÇÃO!\n\n");
        break;
    }
    printf("SUCESSO NA REINICIALIZAÇÃO!\n\n");
    break;
```

Caso o valor for 7 o programa reinicializará a multi pilha:

```
case 7:

    if (mps == NULL) {
        printf("\nMultipilha não foi criada ainda!\n");
        break;
    }

    printf("ATENCAO! OS DADOS INSERIDOS SERÃO PERDIDOS!\n");
    printf("Deseja prosseguir? (0 = NÃO) e (1 = SIM)\n");
    int escolha;
    scanf("%d", &escolha);
    while (escolha != 0 && escolha != 1)
    {
        printf("Digite um valor valido!!!\n\n");
        scanf("%d", &escolha);
    }
    if (escolha == 1)
    {
        mps = destroi(mps);//libera memoria alocada
        mps = NULL;//destroi a estrutura
        criado = 0;
        printf("\nMultipilha destruida com sucesso!\n");
    }else{
        printf("\nCancelando operacao...\n");
    }
    break;
```

E por fim caso o valor for 8 exibiremos um display para o vetor principal onde “_” representa que nenhuma entrada dada pelo usuário foi inserida naquele espaço, caso o usuário venha a inserir algo em alguma pilha esse vetor será atualizado e pode ser consultado para ilustração e melhor compreensão.

TESTE DE MESA DA QUESTÃO 1:

Observação: fizemos testes de várias condições de todos os casos para garantir que a memória esteja sendo manipulada de forma correta e o código opere perfeitamente, então vamos apresentar aqui um teste em que efetuamos as principais funções contidas no nosso código e exibimos prints do terminal para melhor compreensão.

Teste 1 (colisão de topos):
Passo 1: criar o vetor com tamanho 4:

```
1
Insira o tamanho do vetor que deseja criar:
4
Vetor criado com sucesso!
```

```
Printando o vetor atualizado:
_ _ _ _
```

Passo 2: inserir “A”, “B”, e “C” na pilha 1:

```
Printando o vetor atualizado:
A B C _
```

Passo 3: inserir o inteiro 4 na pilha 2:

```
Printando o vetor atualizado:
A B C 4
```

Passo 4: verificar o número de elementos no vetor:

```
5

Numero de elementos inseridos no vetor: 4
```

Passo 5: consultar o topo da pilha 1:

```
4
Consultar o topo da pilha 1 ou 2?
1
Consultando o topo da pilha 1:
Elemento: C
```

Passo 6: consultar o topo da pilha 2:

```
4
Consultar o topo da pilha 1 ou 2?
2
Consultando o topo da pilha 2:
Elemento: 4
```

Passo 7: reinicializar a pilha 2:

```
6

Qual pilha deseja reinicializar (1 ou 2)?
2
SUCESSO NA REINICIALIZAÇÃO!
```

```
Printando o vetor atualizado:
A B C _
```

Passo 8: consultar o número de elementos do vetor para garantir que está sendo alterado

```
5

Numero de elementos inseridos no vetor: 3
```

Passo 9: tentar remover o topo da pilha 2 (dará erro pois acabamos de reiniciá-la):

```
3
Qual pilha (1 ou 2)?
2
A pilha está vazia!!

Erro na remocao!!
```

Passo 10: remover o topo da pilha 1:

```
3
Qual pilha (1 ou 2)?
1
Removido com sucesso!
Item removido: C
```

```

Printando o vetor atualizado:
A B _ _
```

Passo 11: inserir os valores 1 e 2 na pilha 2:

```

Printando o vetor atualizado:
A B 2 1
```

Passo 12: tentar inserir mais um elemento em qualquer pilha (dará erro pois mp->topo2 - mp->topo1 == 1)

```
2
Deseja inserir em qual pilha (1 ou 2)?
1
Deseja inserir um inteiro (Digite 0) ou um caractere (Digite output$ 1)?
1
Insira o caractere que deseja inserir:
C
ERRO! Vetor totalmente preenchido!!

FALHA AO INSERIR NA PILHA!
```

Passo 13: destruir a multi pilha:

```
7
ATENCAO! OS DADOS INSERIDOS SERÃO PERDIDOS!
Deseja prosseguir? (0 = NÃO) e (1 = SIM)
1

Multipilha destruida com sucesso!
```

Ao tentar exibir o vetor:

```
8
Multipilha não foi criada ainda!
```

Passo 14: escolher novamente o tamanho do vetor, mas agora com tamanho 5 (visto que todas as outras funções retornam erro pois a multi pilha não foi criada ainda):

```
1
Insira o tamanho do vetor que deseja criar:
5
Vetor criado com sucesso!
```

```
Printando o vetor atualizado:
- - - - -
```

Passo 15: verificar a quantidade de elementos inseridos no vetor (será 0 pois acabamos de inicializar a multi pilha):

```
5

Numero de elementos inseridos no vetor: 0
```

Questão 2:

O código da questão 2 também foi desenvolvido utilizando VsCode mas foi testado no terminal do Linux utilizando a seguinte linha de comando:

```
gcc -Wall pilha.c arq.c -Larq.h && ./a.out nome_do_arquivo.html
```

Utilizamos uma pilha dinâmica simplesmente encadeada, pois a memória para cada nó será alocada somente conforme o necessário e simplesmente encadeada pois existe uma cadeia de único sentido fruto do ponteiro existente em cada nó apontado para o anterior. Segue abaixo nosso arq.h sem nenhuma grande especificação:

arq.h > ...

```c
#include<stdio.h>
#include<stdlib.h>
#include<string.h>
#include<stdio_ext.h>

//===============MODELO DE DADOS DA APLICACAO=============
typedef struct infoDados{
char str[50];
}info;

//====================MODELO DE DADOS====================
//pilha dinamica simplesmente encadeada
typedef struct nos
{
   info data;
   struct nos *anterior;//unico sentido
}noPilha;

typedef struct descritor
{
    noPilha *topo;
    noPilha *vetor;
    int tam;
}descPilha;

//====================FUNCIONALIDADE DA PILHA====================
descPilha *criaPilha();
int insereNaPilha(descPilha *p, info *novo);
int removeDaPilha(descPilha *p, info *removido);
```

Partindo para o arq.c temos a função criaPilha() que seja a montagem padrão de uma pilha dinâmica simplesmente encadeada:

```c
descPilha *criaPilha()
{
    descPilha *p;
    //verifica a alocacao de memoria inicial para a pilha(1, porque vai variar e acordo com insercoes e remocoes)
    if ((p = malloc(1 * sizeof(descPilha))) == NULL)
    {
        printf("ERRO! Nao foi possível criar a pilha!\n\n");
        free(p);
        p = NULL;
        return NULL;
    }
    else
    {
        printf("Sucesso na criacao da pilha!\n\n");
        //verifica a alocacao de memoria inicial para o vetor onde a pilha estara inserida(1, porque vai variar e acordo com insercoes e remocoes)
        if ((p->vetor = malloc(1 * sizeof(noPilha))) == NULL)
        {
            printf("ERRO! Nao foi possivel inicializar o vetor!\n\n");
            free(p->vetor);
            free(p);
            p->vetor = NULL;
            p = NULL;
            return NULL;
        }
        else
        {
            printf("Sucesso na inicializacao do vetor!!\n\n");
            //inicializacao dos campos da pilha
            p->topo = &(p->vetor[0]);
            p->vetor[0].anterior = NULL;
            p->vetor[0].data.str[0] = '\0';
            p->tam = 0;
        }
    }
    return p;
}
```

A função insereNaPilha(), depois de inúmeros testes concluímos que a melhor forma de salvar a string na pilha é através de da chamada strcpy abaixo:

```c
int insereNaPilha(descPilha *p, info *novo)
{
    if (p == NULL)//verifica a existencia da pilha
    {
        printf("Pilha não criada!!\n\n");
        return 0;
    }
    if (p->vetor == NULL)//verifica a existencia do vetor
    {
        printf("Vetor não inicializado!\n\n");
        return 0;
    }

    //realloca a memoria do vetor para receber a proxima string
    if ((p->vetor = realloc(p->vetor, ((p->tam) + 1) * sizeof(noPilha))) == NULL)
    {
        printf("PROBLEMA NA REALOCACAO DE MEMORIA!!\n\n");
        return 0;
    }

    //insere a string no vetor
    strcpy(p->vetor[p->tam].data.str, novo->str);

    //redefine o topo da pilha
    p->topo = &(p->vetor[p->tam]);

    if (p->tam != 0)//encadeia o novo termo com o seu antecessor
    {
        p->topo->anterior = &(p->vetor[p->tam - 1]);
    }

    p->tam += 1;

    return 1;
}
```

A função removeDaPilha() segue abaixo comentada com o raciocínio de cada ação:

```c
int removeDaPilha(descPilha *p, info *removido)
{
    if (p == NULL)//verifica a existencia da pilha
    {
        printf("Pilha não criada!\n\n");
        return 0;
    }
    if (p->vetor == NULL)//verifica a existencia de vetor
    {
        printf("Vetor nao inicializado!\n\n");
        return 0;
    }

    //realloca a memoria para 1 termo a menos (o vetor fica com a tamanho da quantia de termos exata que ele tem)
    if ((p->vetor = realloc(p->vetor, (p->tam) * sizeof(noPilha))) == NULL)
    {
        printf("ERRO NA REALOCACAO DE MEMORIA!\n\n");
        return 0;
    }

//define a string removida
    (*removido) = p->topo->data;
    //redefine a string excluida
    p->vetor[p->tam-1].data.str[0] = '\0';
    //redefine o ponteiro da string excluida que aponta para seu antecessor como NULL "quebrando" o encadeamento entre eles
    p->topo->anterior = NULL;
    //redefine o tamanho
    p->tam -= 1;
    //redefine o topo da pilha
    p->topo = &(p->vetor[p->tam-1]);
    return 1;
}
```

Partindo para o arquivo pilha.c é onde vamos detalhar mais os passos e explicar os pontos cruciais do nosso código. Após as inicializações e declarações fundamentais para o programa temos o seguinte trecho:

```c
76      //realizacao da leitura do arquivo ate que o fgets retorne NULL (fim do arquivo)
77      printf(YELLOW "--- INICIO DA ANALISE DO HTML ---\n" RESET);
78      while (fgets(c, 5000, file) != NULL)
79      {
80          //concatena a string 'c' em 'html' verificando a existencia de um valor de retorno (no caso um ponteiro)
81          //se não existir eh porque deu erro na funcao
82          strcat(html, c);
83
84          printf("Linha da html: %s\n", c);
85      }
86      printf(YELLOW "--- FIM DA ANALISE DO HTML ---\n" RESET);
```

Na imagem acima vemos o armazenamento das strings contidas no arquivo html para a variável html através da função strcat.

Mais abaixo temos um elemento crucial do nosso desenvolvimento:

```c
90      token1 = strtok(html, v1);//primeiro token '>' e ' '
91
92          //passagem pelos outros tokens e verificacao se houve um erro de aninhamento
93          printf("\nExibição da evolução da pilha:\n");
94          while (token1 != NULL && erro == 0)
95          {
96              //mostrando a pilha
97              ajuda = pilha->topo;
98              printf(BLUE "\nPilha:\n" RESET);
99              for (int i = 0; i < pilha->tam; i++)
100             {
101                 printf("%s\n", ajuda->data.str);
102                 ajuda = ajuda->anterior;
103             }
```

A análise do primeiro token, que no caso é o fechamento de tag ‘>’ e espaços vazios, com isso entramos no while e o programa só irá sair após finalizar a leitura de toda a variável html. O código continua verificando se cada token contém uma abertura de tag com ‘<’. Se

for encontrado a abertura ele faz uma verificação para determinar o tipo de tag (normal, um comentário ou do tipo DocTYPE):

```
//verificacao se há a abertura de tags ate o primeiro ' ' ou primeiro '>'
for (int i = 0; i <= (int)strlen(token1); i++)
{
    if (token1[i] == v3[0])//ve se tem abertura de tag
    {
        if (token1[i + 1] == v4[0])//ve se eh um comentario ou !DOCTYPE
        {
            sFecha = 1;
        }
        else
        {
            indice = i;
            aberturaParaTag = 1;
        }
    }
}
```

Após esse trecho teremos a análise da tag a partir da aberta dela e salvaremos essa string (caso tudo estiver ok na tag) no vetor strAux[] para podermos fazer análises e manipulações futuras dessa string como comparação com outra tag e remoção. Nesse trecho também fazemos a verificação das tags que não vamos armazenar:

```
int j = 0; //auxiliar para montagem da strAux e da strParaExcluir
if (aberturaParaTag == 1)//verifica se houve abertura de tag
{
    //monta a strAux para auxiliar a analisar a string posteriormente
    for (int i = indice + 1; i <= (int)strlen(token1); i++)
    {
        strAux[j] = token1[i];
        j++;
    }

    //verificacoes se eh uma tag especifica que nao tem fechamento (nao serao inseridas na pilha)
    if (strcmp(strAux, nFecha1) == 0)
    {
        printf("Encontrou: TAG DO TIPO <img>! (Nao iremos adicionar)\n\n");
        sFecha = 1;
    }
    if (strcmp(strAux, nFecha2) == 0)
    {
        printf("Encontrou: TAG DO TIPO <br>!! (Nao iremos adicionar)\n\n");
        sFecha = 1;
    }
    if (strcmp(strAux, nFecha3) == 0)
    {
        printf("Encontrou: TAG DO TIPO <input>!! (Nao iremos adicionar)\n\n");
        sFecha = 1;
    }
    if (strcmp(strAux, nFecha4) == 0)
    {
        printf("Encontrou: TAG DO TIPO <frame>!! (Nao iremos adicionar)\n\n");
        sFecha = 1;
    }
}
```

Logo a baixo efetuamos a análise de tags com abertura e de fechamento e comparamos elas com as já armazenadas na pilha e fazermos sua exclusão da pilha caso tudo estiver ok significando que houve uma abertura de fechamento correto dessa tag ao longo do html:

```
if (aberturaParaTag == 1 && sFecha == 0)//verifica se houve abertura de tag e se a tag tem fechamento
{

    if (strAux[0] == v2[0])//verificacao se a tag tem '/' o que indica ser uma tag de fechamento
    {
        fecharTag = 1;
    }
    if (fecharTag == 1)//verifica se eh uma tag de fechamento
    {
        j = 0;
        //montagem do strParaExcluir
        for (int i = 1; i <= (int)strlen(strAux); i++)
        {
            strParaExcluir[j] = strAux[i];
            j++;
        }

        //comparacao entre o topo da pilha e a tag que eh para ser fechada, verificando se sao iguais
        if (strcmp(pilha->topo->data.str, strParaExcluir) == 0)
        {

            //executa a remocao do elemento da pilha verificando se deu certo
            if (removeDaPilha(pilha, &removido) == 1)
            {
                printf(RED "\n(Atualizacao da pilha) String removida (encontrou fechamento): "RESET);
                printf("%s", removido.str);

                //redefinicao da strAux
                strcpy(strAux, pilha->topo->data.str);
            }
            else
            {
                printf(RED "FALHA NA REMOCAO\n\n" RESET);
            }
        }
        else
        {
            erro = 1;
```

Logo abaixo do erro = 1, exibimos a mensagem que o professor solicitou onde devemos mostrar ao usuário os erros e correções dos mesmo:

```
        //mensagem de erro de aninhamento
        printf("\nERRO! Esperado </%s>, recebido <%s>.\n Sugestao: Fechar a tag <%s> com </%s> ou abrir uma tag <%s> antes de fechá-la!\n\n",
        pilha->topo->data.str, strAux, pilha->topo->data.str, pilha->topo->data.str, strParaExcluir);
    }
```

Caso a tag tenha seja de abertura, ou seja, seja do tipo <body> por exemplo e é a primeira vez que foi encontrada no html então iremos adicioná-la no topo da pilha para verificar o restante do html dentro do while esperando encontrar sua correspondente tag de fechamento:

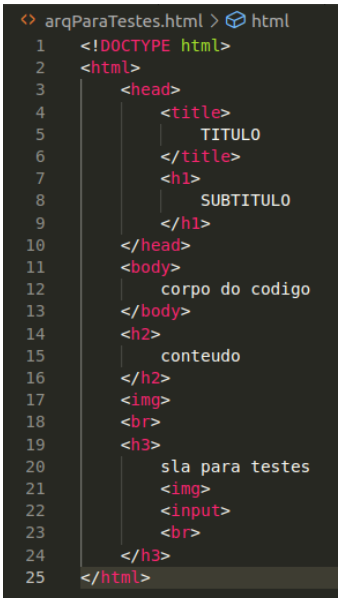
```
else
{
    //definindo o 'aux' para insercao na pilha com verificacao
    strcpy(aux.str, strAux);


        //insercao na pilha com verificacao
        if (insereNaPilha(pilha, &aux) == 1)
        {
            printf(GREEN "\nSUCESSO NA INSERCAO (ENCONTROU ABERTURA)! ATUALIZANDO PILHA:" RESET);
        }
        else
        {
            printf(RED "FALHA NA INSERCAO!" RESET);
        }

}
fecharTag = 0;//reinicializacao da 'fecharTag'
```

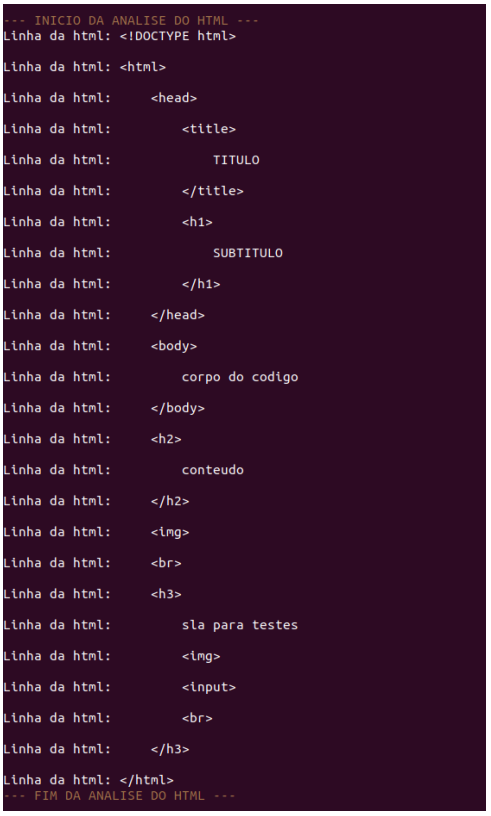
PRIMEIRO TESTE DE MESA DA QUESTÃO 2:

Para esse teste iremos compilar o arquivo arqParaTestes.html que foi escrito de com tags de abertura e fechamento perfeitamente alinhadas e também contém tags que devemos ignorar e não levar em consideração na inserção da pilha, importante salientar que nosso código é capaz de desconsiderar textos inseridos entre tags por conta dos tokens:

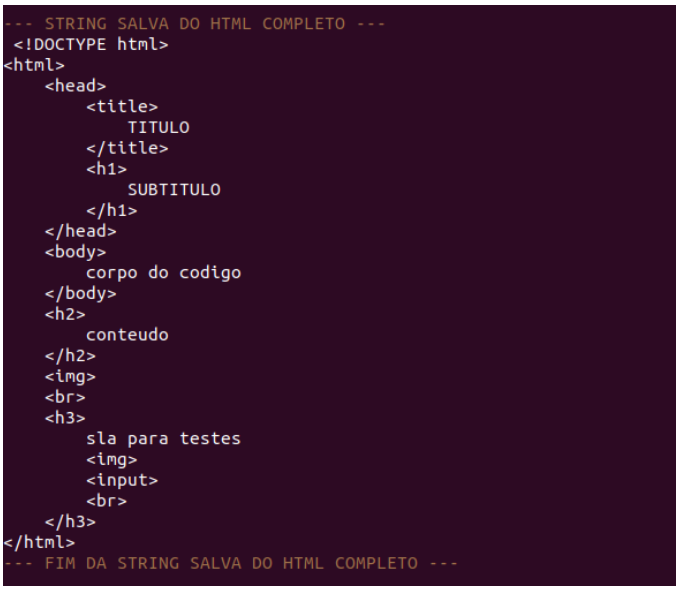


Ao compilar os arquivos através do comando:
gcc -Wall pilha.c arq.c -Larq.h && ./a.out arqParaTestes.html

Teremos sempre um print do que estamos captando de cada linha do arquivo html compilado como esse a seguir:



E também sempre será exibido para o usuário a forma como a nossa variável html (a string que será analisada pelos tokens e segmentada para ser empilhada na pilha):



Após essas saidas inicias o usuario terá contado com o estado da pilha atravez de prints consecutivos (infelizmente por conta do laco do while alguns prints do estado da pilha sao repetidos, daria muito trabalho eliminar essas repeticoes entao mantivemos no codigo final):

```
Exibição da evolução da pilha:

Pilha:

Pilha:

Pilha:

SUCESSO NA INSERCAO (ENCONTROU ABERTURA)! ATUALIZANDO PILHA:
Pilha:
html

Pilha:
html

SUCESSO NA INSERCAO (ENCONTROU ABERTURA)! ATUALIZANDO PILHA:
Pilha:
head
html

Pilha:
head
html

SUCESSO NA INSERCAO (ENCONTROU ABERTURA)! ATUALIZANDO PILHA:
Pilha:
title
head
html
```

Acima vemos que a pilha se encontra vazia por algumas verificações e ao encontrar a tag <html> verifica a mesma e conclui que se trata de uma tag de abertura, logo aciona a string no topo da pilha e atualiza na saída, e após alguns passo encontra a abertura da tag head e head adiciona as mesmas também.

Após alguns passos encontramos a tag </title> e o programa reconhecerá ela como uma tag de fechamento portanto como o topo atual é title teremos a seguinte modificação na pilha (exclusão do topo title pois encontrou o fechamento):

```
(Atualizacao da pilha) String removida (encontrou fechamento): title
Pilha:
head
html
```

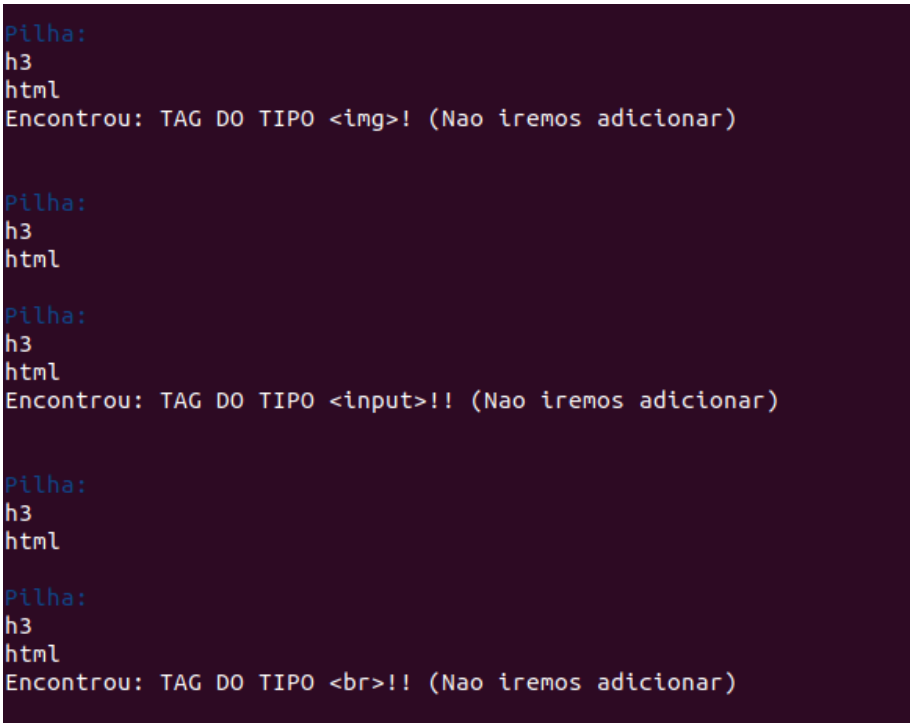
Seguindo a analise do html será encontrado <h1>:

```
SUCESSO NA INSERCAO (ENCONTROU ABERTURA)! ATUALIZANDO PILHA:
Pilha:
h1
head
html
```

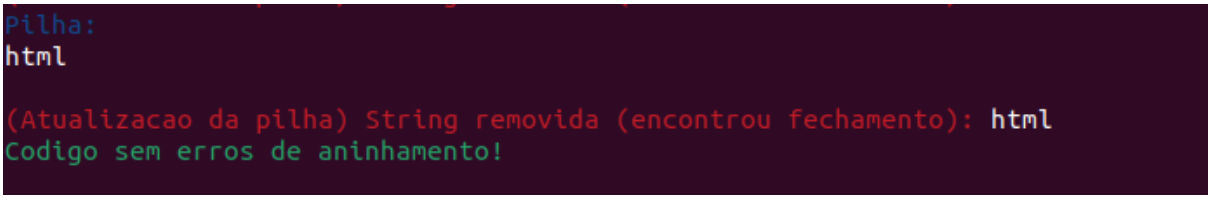
Em seguida encontrará </h1> e teremos:

```
(Atualizacao da pilha) String removida (encontrou fechamento): h1
Pilha:
head
html
```

E o código seguirá funcionando para as demais tags, inclusive as que devemos desconsiderar como essas a seguir:



E por fim o programa apos analisar todas as tags encontrará a tag </html> e teremos:



SEGUNDO TESTE DE MESA DA QUESTÃO 2:

Nesse teste compilamos o arquivo arqParaTestesErrados.html usando o comando:
gcc -Wall pilha.c arq.c -Larq.h && ./a.out arqParaTestesErrados.html

O arquivo possui a seguinte estrutura:

```
arqParaTestesErrados.html > ...
1   <!DOCTYPE html>
2   <html>
3       <head>
4           <title>
5               TITULO
6           </title>
7           <h1>
8               SUBTITULO
9           </h1>
10      </head>
11      <body>
12          corpo do codigo
13      </body>
14      <h2>
15          conteudo
16      </h2>
17      <img>
18      <br>
19      <h3>
20          sla para testes
21          <img>
22          <input>
23          </h3>
24          <br>
25      </h3>
26
27  </html>
```

Ao compilarmos os arquivos teremos a exibição do html assim como o empilhamento das strings de tags até chegar no erro do código que é na linha 25 com um duplo fechamento da tag h3, portanto a saída será:

```
Pilha:
html

ERRO! Esperado </html>, recebido </h3>.
 Sugestao: Fechar a tag <html> com </html> ou abrir uma tag <h3> antes de fechá-la!

Encerrando a execucao do programa...
```

TERCEIRO TESTE DE MESA DA QUESTÃO 2:

Manipulamos o arqParaTestesErrados.html e dessa vez colocamos a abertura de <h1> entre a tag title e </h1> fora do fechamento da tag title como segue a baixo:

```
arqParaTestesErrados.html > html > head > title
1   <!DOCTYPE html>
2   <html>
3       <head>
4           <title>
5               TITULO
6           <h1>
7           </title>
8               SUBTITULO
9           </h1>
10      </head>
11      <body>
12          corpo do codigo
13      </body>
14      <h2>
15          conteudo
16      </h2>
17      <img>
18      <br>
19      <h3>
20          sla para testes
21          <img>
22          <input>
23          </h3>
24          <br>
25      </h3>
26
27  </html>
```

Dessa forma teremos a seguinte saída final:

```
Pilha:
h1
title
head
html

ERRO! Esperado </h1>, recebido </title>.
 Sugestao: Fechar a tag <h1> com </h1> ou abrir uma tag <title> antes de fechá-la!

Encerrando a execucao do programa...
```