# TP1-ED - Servidor de Emails

## Lucas Santana do Carmo Sacramento - 2021031700.

### Introdução: Contém a apresentação do contexto, problema, e qual solução será empregada.

Neste trabalho Prático disponibilizado pela matéria de Estrutura de Dados, nos coloca na pele de um programador de um servidor de e-mails, no qual o usuário do TAD terá a liberdade para cadastrar um usuário, remover, entregar uma mensagem e consultar a caixa de entrada pra determinado usuário.

As mensagens serão exibidas seguindo uma ordem de prioridade, na qual pode variar de 0 a 9 e seguindo essa premissa uma mensagem com mesma ordem não pode sobrepor a outra não entrar em sua frente em uma possível consulta.

Além disso, deve ser informado um erro caso funções sejam chamadas sem uma alocação previa. Ex: consultar usuário que não foi cadastrado, dessa forma, deve exibir um erro de usuário inexistente e assim por diante.

Para resolver este problema, não podemos utilizar alocação estática, uma vez que só o usuario do TAD tem acesso ao numero de usuario que serão cadastrados, então, dessa forma, será feita atraves de alocação dinamica, com isso, a utilização de listas encadeadas será de fundamental importancia, uma vez que o limite da sua alocação será apenas dada ao uso de meemoria que será alocado ao programa.

Uma vez que foi definido o metodo para a implementação do programa, agora devemos definir quais TADs usaremos para a resolução do problema.

Inicialmente o contrato foi idealizado com 4 TADs: Mensagem, CaixaEntrada, Usuario, ListaUsuario. Dessa forma, será utilizada Mensagem e Usuario como objetos responsaveis por amazenar os atributos e os TADs CaixaEntrada e ListaUsuario, como objetos responsaveis por implementar a lista encadeada de cada um dos objetos.

trabalho pratico 1 Estrutura de Dados

# Método

O Trabalho prático está dividido em quatro TADs, sendo eles:

Email, CaixaEntrada, Usuario, ListaUsuario, com as seguintes funcionalidades.

## Contrato dos TADs

### Email

O Tad Email conta com os atributos:

Prioridade: variavel do tipo inteiro que será usada posteriormente para fazer a inserção na lista pela ordem de prioridade

Msg: Variável do tipo string que será armazenado o conteúdo contido no email.

next: Ponteiro do tipo mensagem que será utilizado no TAD CaixaEntrada como artificio pra percorrer a lista encadeada, sendo next uma variavel do tipo ponteiro que irá armazenar o endereço de memoria para um outro objeto do tipo mensagem.

Construtor Mensagem: Função que não possui paramentro, nem saida, ele será utilizada como um construtor padrão inicialização em outras classes por meio de composição.

Sobrecarga do construtor Mensagem: Construtor que é declarado com dois paramentros: prioridade e mensagem. Nele é atribuido os valores para os atributos, além de declarar o ponteiro next como nulo.

Destrutor Mensagem: Declarado de forma virtual o destrutor para o objeto mensagem não possui nenhuma função em especifico.

### CaixaEntrada

Atributos do TAD CaixaEntrada:

head: ponteiro do tipo Mensagem que será utilizado como cabeça pra a lista encadeada de e-mails

Construtor CaixaEntradada: O construtor da classe CaixaEntrada apenas atribui o ponteiro de Mensagem head, como nulo.

Sobrecarga do construtor CaixaEntrada: a primeira vez que o construtor da classe CaixaEntrada é chamada, ele receberá um objeto do tipo Mensagem e será atribuido como cabeça da lista encadeada.

Insere Prioridade: Método que recebe um objeto do tipo mensagem e que realiza a seguinte operação.

Ao ser chamada, ela irá criar uma nova mensagem, a mesma que foi recebida como parametro da função, após isso, irá ser inicializado dois ponteiros, chamados auxiliar1 e auxiliar2 do tipo Mensagem que ambos são apontados para a cabeça da lista.

O objeto Nova\_mensagem que foi criado, tem seu ponteiro next inicializado como nulo.

Após essas operações um while é iniciado, ele irá fazer duas comparações, checar se o auxiliar2 é um valor diferente de nulo e se o atributo prioridade do auxiliar é maior ou igual ao atributo prioridade da nova mensagem.

caso a afirmação for verdadeira:

O auxiliar1 irá receber o endereço de auxiliar2 que está iterando a lista e o auxiliar2 recebe um ponteiro para o proximo email.

caso a afirmação for falsa:

para os dois casos ele será testado na estrutura de condição abaixo do while, porem é checado para caso a nova mensagem for maior que o primeiro elemento da lista. Neste caso, o auxiliar 1 que possui um ponteiro pra a cabeça aponta para a nova mensagem transformando ela na cabeça da lista. Analogo a isso, o else da estrutura condicional insere a mensagem sequencialmente.

MostrarPrioridade: Metodo do tipo void que não possui nenhum parametro para ser inicializado. O metodo iniciará duas variaveis do tipo ponteiro Mensagem, mensagens e temporario, ambas sendo apontadas para a cabeça da lista de emails.

através de uma estrutura de condição, primeiro é checado se a cabeça da lista é nula, caso seja verdadeiro, é retornado o erro de que a caixa de entrada está vazia, caso contrário, será exibida a mensagem atraves do ponteiro mensagens que foi declarado no escopo da função, a variavel temporaria que tambem está apontando para head, aponta para o segundo elemento da lista, o elemento que está na cabeça da lista (mensagem com maior prioridade e com ordem de chegada) é desalocado. Dessa forma o elemento cabeça recebe a variavel temporada que no momento está apontando para o elemento seguinte da lista.

DesalocaMensagens: Metodo do tipo vazio que não possui nenhum parametro. Ele é responsável por desalocar todos os objetos do tipo Mensagens ao final da execução do programa, para isso, inicialmente são alocado uma variavel ponteiro do tipo Mensagem que será usada para iterar a caixa de entrada e desalocar as mensagens.

apos a inicialização do ponteiro mensagem e passado uma estrutura condicional que checa, caso a caixa de entrada do usuario já está vazia ou não com a expressão: se cabeça for nula. Caso seja, apenas é deletada a cabeça da lista, caso contrario, o ponteiro declarada no inicio do metodo, recebe o elemento posterior a head e inicializa um while até a variavel mensagem ser totalmente percorrida, dentro do while ó objeto alocado no head é passado para o ponteiro mensagem que em seguida é deletado da memoria, apos isso, a variavel mensagem recebe novamente o proximo elemento e Lassim é iterado até a todas as mensagens serem totalmente desalocadas.

Destrutor CaixaEntrada: Destrutor virtual que recebe o metodo DesalocaMensagens como descrita a cima e após isso, desaloca o elemento que se encontra na cabeça da lista.

### Usuario

O TAD Usuario além de possuir um atributo do tipo id e um ponteiro do mesmo tipo para a implementação da lista de usuario, também, atraves de composição recebe o TAD CaixaEntrada. Dessa forma, cada Usuario irá possuir uma caixa de entrada separadamente com o atributo Id para diferencia-los entre si.

Atributos do TAD Usuario

Id: atributo do tipo inteiro que será utilizada para identificar o usuario no momento do cadastro em uma fila encadeada do tipo Usuario.

Emails: Atributo do tipo CaixaEntrada, uma lista encadeada de emails.

next: ponteiro do tipo Usuario que será o nó da lista encadeada de usuarios.

Usuario: Construtor padrão do objeto usuario

Sobrecarga Usuario: Construtor sobrecarregado de usuario que recebe como parametro o id do usuario e atribui o valor do parametro ao atributo id.

RecebeMensagens: Metodo do tipo void que recebe como parametro um objeto do tipo mensagem que instancia um metodo do TAD Caixa de entrada, chamadando a função Insere\_Prioridade

ExibeMensagens: Metodo do tipo void que possui uma instancia caixa de entrada que exibe uma mensagem instanciando um metodo da composição da caixa de entrada.

DesalocaMensagens: Metodo do tipo void que não recebe parametro e que instancia um metodo do TAD caixa de entrada DesalocaMensagens.

Destrutor Usuario: Destrutor virtual que não possui nenhuma implementação.

### ListaUsuario

O TAD ListaUsuario possui a responsabilidade de coordenar todos os outros 3 TAD criados.

Como uma lista de Usuario que consequentemente cada um deles possui uma caixa de entrada, esse será o contrato com o maior numero de instancias em suas aplicações

Atributos do TAD ListaUsuario:

head: Ponteiro do tipo usuario que será responsável por inicializar a lista encadeada de ususarios.

Construtor de ListaUsuario: construtor padrão que não recebe parametros e que possui como função declarar head como um ponteiro nulo.

CadastraUsuario: Método do tipo void que possui como parametro um valor do tipo int que chamaremos de id. Após a chamada desse metodo será inicializado um ponteiro do tipo Usuario chamado auxiliar que receberá o endereço de head, após essa declaração head será submetido a uma estrutura condicional. Se caso head for nula, assume-se que a lista está vazia e dessa forma é cadastrado um usuário, dessa forma, head irá receber o endereço de um objeto do tipo Usuario que terá como parametro o valor inteiro id, retornando assim uma string informando que a conta foi devidamente cadastrada. Caso head não for nula, irá ser inicializada uma estrutura de repetição que vai comparar o id que foi passado com o id de todos os usuario que ja foram devidamente cadastrados e se caso houver algum usuário com mesmo id do que foi passado, é retornado um erro e a função é encerrada, O while percorre até a penultima posição, mas não valida para a ultima, dessa forma ele encerra a execução e checa se a ultima posição possui o mesmo id do que foi passado por parametro, caso não tenha sido o auxiliar next que agora se encontra na ultima posição recebe o endereço de um novo usuario com o id passado posteriormente.

MostrarCaixaEntrada: Método do tipo void que recebe como parametro um inteiro de nome id.

Nessa função head é passada para uma variavel do tipo ponteiro Usuario que será iterada em um while com uma condição que se o id passado por parametro for o mesmo do usuario que está sendo iterado, esse mesmo usuário irá instanciar uma função do objeto usuario que ExibeMensagens que por consequencia instancia uma função do objeto caixa de entrada para exibir a mensagem com maior prioridade. Caso o id que foi passado não estiver presente na lista, é exibido um erro informando que o usuario não existe e assim finalizando a execução da função.

EnviaMensagem: Metodo do tipo void que recebe como parametro um atributo do tipo inteiro id e um objeto do tipo Mensagem.

Nessa função será feita uma implementação analoga ao que foi feito na função que mostra a caixa de entrada, inicializando um ponteiro para head, iterando sobre ele, procurando um id semelhando ao que foi passado por parametro e se caso for encontrado, instanciar o metodo RecebeMensagem passando como parametro o objeto mensagem e exibindo uma string informando que a mensagem foi entregue, caso contrario não será enviada a mensagem e será impressa a mensagem de que o usuario não existe.

ProcuraUsuario: Metodo do tipo booleano que será importante para encontrar se um usuario está presente na lista ou não.

Ela recebe como paramentro um id e itera a lista encadeada procurando o usuario de mesmo id, caso não encontre, ela retornará 0, caso encontre, retornará 1. Essa função será bem importante para a função que será descrita logo em seguida, o metodo removeUsuario.

RemoveUsuario: Metodo do tipo void que recebe como parametro o id do usuario que é desejado a remoção.

Nesta funçao são inicializados dois ponteiro do tipo Usuario, um recebendo a cabeça da lista e outro vazio que será utilizado como temporario para a realização da remoção do usuario na lista. Após isso a função será submetida a uma estutura de condição que irá usar a função procura usuario para validar a existencia ou não do usuario na lista, caso não existir, será impressa uma mensagem da não existencia e encerrara o programa, caso contrário será inicializado um while que percorerá toda a lista, dentro do while ha 3 condições, se caso o id passado estiver em um usuario que se encontra na cabeça, a cabeça irá receber a o elemento seguinte e itera novamente o while. Se o auxiliar next for nulo, é encerrado o while e a mensagem de que usuario é inexistente é exibida. E por ultimo, se caso for encontrado o usuario, um segundo auxiliar recebe o usuario, o primeiro auxiliar que está iteradno a lista recebe o usuario seguinte para a ligação da lista e não deletar um usuario no meio da lista e por ultimo deleta o usuario do segundo auxiliar, finalizando o ciclo, removendo o usuario com base no seu id.

Limpa: Metodo do tipo void que não recebe nenhum parametro tem por objetivo desalocar todos os usuarios restantes na lista encadeada.

Através de um ponteiro auxiliar que é recebido o segundo elemento da lista, é passado por um for desaloca todos os usuario restantes, sobrendo apenas o elemento head.

Destrutor de ListaUsuario: Esse destrutor ira checar sempre se a cabeça da lista possui conteudo, caso exista, chama a função limpa, exemplificada a cima e por ultimo deleta o elemento da cabeça que foi deixado em aberto quando chamada a função Limpa, e assim é finalizada a execução do programa.

### Main

O Main foi organizado de forma que todos os elementos necessarios para a funcinalidade dos filtros esteja contido nele.

Iniciando por uma função chamada isNumber que recebe como parametro um string e itera a string, caso encontre alguma palavra retorna falso, caso contrário retorna o valor true, certificando que aquela string só possui valores numericos, isso vai ser muito importante, uma vez que o arquivo de texto que será passado atráves do terminal admite apenas valores do tipo string.

Atraves de argumentos, será possivel passar um arquivo de texto como entrada para a realização do tratamento, ao declarar as variaveis que serão utilizadas para a manipulação do arquivo destaca-se algumas do tipo string que recebem palavras chaves como: "CONSULTA", "CADASTRA", "ENTREGA" e "REMOVE", após a declaração dessas variaveis, será declarado o objeto do tipo ListaUsuario para a manipulação do lista encadeada, e a abertura do arquivo de texto. Será utilizado um while que possui criterio de parada o fim do arquivo, dessa forma iterando cada linha do arquivo e armazenando em uma variavel do tipo string que será manipulada, após o recebimento de cada linha pela variavel do tipo string é utilizada a função find, guardando seu conteudo em uma variavel do tipo size\_t, o valor retornado para esse tipo de string informa o numero de ocorrencias do que foi passado a ele, nesse caso a instancia de uma caida de caracteres em uma string, caso o valor seja diferente do atributo string::npos, significa que foi encontrado um instancia diferente, logo como é usado o validador diferente, a palavra foi encontrada. Sendo assim um estrutura de condição mapeia as 4 alternativas, para todas as opções serão inicializada duas variaveis do tipo inteiro pos e tam, eles serão limitadores para delimitarmos um intervalo pra selecionar um subconjunto de strings. Caso a instancia cadastra seja selecionada, vamos ignorar a primeira palavra, uma vez que é a propria palavra cadastra e a partir de um delimitador que nesse caso é o espaço, pegaremos o proximo valor imediatamente apos o espaço, uma string chamada digito receberá o valor numero em formato de string e apartir dai utilizaremos o função isNumber para checar que todos os elementos do tipo string são numericos, fazendo assim a conversão para int atraves da função stoi e incluiremos na Lista um novo usuario com esse valor que foi passado no arquivo de texto. Caso a função isNumber retorne o valor false, significa que o valor passado nao é numerico, então dessa forma é lançado um assert e o codigo é abortado de imediato.

Para os casos CONSULTA e REMOVE, funcionam da mesma forma, pegando o valor da string, checando se é numerica e convertendo caso seja verdadeira ou lançando uma asserção caso seja falsa e apos isso Chamando uma instancia para Lista para cada funcionalidade respectivamente.

Por ultimo tem-se a função ENTREGA que possui os parametros id do destinatário, prioridade da mensagem e a mensagem em si.

Utilizando os mesmos conceitos para as opções anteriores, a unica diferença para a função entrega é que ele deve checar se uma cadeia de caracteres é numerica ou não para o id e para a prioridade da mensagem e lançando uma asserção, caso não esteja de acordo com o padrão, após isso, pos irá receber o retorno da função find, ela retornará todas as mensagens que possui no final a expressão "FIM", dessa forma pegando toda a mensagem até o seu ponto final, não havendo a necessidade de iterar manualmente para encontrar o momento em que um ponto final é utilizado, dessa forma é criado o objeto do tipo mensagem com a prioridade e a mensagem e é chamada a lista com a instancia enviaMensagem que recebe como parametro o id do usuario e o objeto mensagem.

Após isso é fechado o arquivo e logo em seguida a Lista de USuarios é retirada da memoria, caracterizando assim o fim da execução do programa.

# Análise de Complexidade:

Para uma boa analise de complexidade de um programa, deve ser feita função por função e após isso, avaliar todas as operações nela contida para dessa forma avaliar o programa como um todo.

Implementações que seguem o modelo de Lista encadeada, possui uma boa eficiencia, uma vez que é possivel acessar qualquer elemento da lista em O(1), pórem dado que o programa exige um determinado filtro, funções que poderiam ser O(1), acabam se tornando O(n), essas funções serão descritas logo abaixo.

### Mensagem

O Tad Mensagem possui no total 3 metodos e todos com chamadas O(1), apenas com declarações de variaveis.

Mensagem() = O(1)

Mensagem(int prioridade, string mensagem) = O(1)

~Mensagem() = O(1)

### CaixaEntrada

CaixaEntrada() = O(1)

CaixaEntrada(Objeto do tipo Mensagem) = O(1)

Insere\_Prioridade(Objeto do tipo Mensagem) = melhor caso O(1), pior caso O(n)

mostrarPrioridade() = O(1)

DesalocaMensagens() = melhor caso O(1), pior caso O(n)

~CaixaEntrada() = melhor caso O(1), pior caso O(n)

Naturalmente a complexidade do TAD será maior uma vez que ela é responsável por alocar a lista encadeada do objeto mensagem. Dessa forma, funções que possuem inserção podem variar seu custo a depender da ordem de qual elemento tenha sido inserido.

Função Insere\_prioridade:

Melhor caso:

Todos os elementos que forem inseridos possuirem uma prioridade crescente, dessa forma todas as novas mensagens serão inseridas na cabeça, o que transforma a função em O(1)

Pior Caso:

Qualquer outro caso que a nova mensagem possui um valor prioritário menor ou igual a mensagem inicial, dessa forma tendo que fazer o while ser percorrido para a inserção.

Função DesalocaMensagens:

Melhor caso:

Caso aja apenas um elemento na lista, assim deletando apenas o elemento da cabeça em O(1).

Pior caso:

Houver mais de um elemento na lista, fazendo assim um while ser percorrido para desalocar as mensagens presentes na lista encadeada.

~CaixaEntrada:

Melhor caso:

Exemplo parecido com o caso DesalocaMensagens, uma vez que ele instancia a função, sendo O(1), caso exista apenas uma Mensagem.

Pior caso:

Caso aja mais de uma mensagem, tendo que instanciar a função DesalocaMensagens que possui um while que percorre a lista para desalocar os elementos presentes nela.

### Usuario

Usuario() = O(1)

Usuario(inteiro id) = O(1)

RecebeMensagem(Objeto do tipo Mensagem) = melhor caso O(1), pior caso O(n)

ExibeMensagens() = O(1)

DesalocaMensagens() = melhor caso O(1), pior caso O(n)

~Usuario() = O(1)

RecebeMensagem:

Função que apenas instancia a função Insere\_Prioridade do TAD CaixaEntrada, logo a explicação para ela é a mesma para a função descrita no TAD.

ExibeMensagens:

Função que instancia uma função que apenas exibe o elemento head de CaixaEntrada, logo essa operação continua sendo O(1).

DesalocaMensagens:

Função com a mesma função descrita anteriormente com o unico adendo de que ela está sendo chamada em usuario que vai ser utilizada em ListaUsuario para desalocar a caixa de entrada de cada usuario cadastrado antes de desalocar o proprio usuario.

### ListaUsuario

ListaUsuario() = O(1)

cadastroUsuario(inteiro id) = melhor caso O(1), pior caso O(n)

mostrarCaixaEntradada(inteiro id) = O(n)

enviaMensagem(inteiro id, Objeto do tipo Mensagem) = O(n)

procuraUsuario(inteiro id) = melhor caso O(1), pior caso O(n)

removerUsuario(inteiro id) = O(n)

Limpa() = O(n)

~ListaUsuario() = melhor caso O(1), pior caso O(n)

cadastroUsuario:

Melhor caso:

O melhor caso será alcançado apenas quando for inserido o primeiro usuario.

Pior caso:

Para qualquer usuario que for cadastrado após o primeiro será rodado um while para checar se ele está presente, logo o pior caso sempre será acionado.

mostrarCaixaEntradada:

Apesar da função que exibe mensagem ser O(1), ela sempre estará sendo dominada por uma função que será O(n), uma vez que ela irá percorrer o while para encontrar o usuario com id compativel com aquele que será exibido.

enviaMensagem:

função que tambem percorre um while para encontrar um usuario pra chamar a instancia de usuario para enviar mensagem.

procuraUsuario:

Função que obrigatoriamente irá percorrer a lista para retornar o booleano, porém pode ser O(1) se o elemento de interesse estiver na cabeça da lista.

removerUsuario:

Função O(n) pelo fato de ter que percorrer dois while, um para checar se o usuario está presente na lista e outro pra apagar ele, logo sua complexidade linear se deve ao fato das comparações de um parametro externo com um atributo do o objeto na lista.

Limpa:

Função O(n), pois possui um while que é obrigatoriamente rodado para deletar todos os objetos do tipo Usuario da lista.

~ListaUsuario:

Melhor caso:

houver apenas um usuario, tendo assim o papel de deletar apenas a cabeça da lista

Pior caso:

Caso aja mais de um elemento na lista é preciso rodar um while para desalocar todos os elementos.

Visão geral:

Dado que para estimar o tempo de execução e o limite assintotico de um programa, devemos analisar função por função, chega-se a conclusão de que o programa possui um limite assintotico O(n).

### Estratégias de Robustez

Como enfatizado no Main a estrategia de Robustez a ser utilizada foram os asserts para valores que são discrepantes daqueles que passado no enunciado ou para valores que não são numericos. A utilização da biblioteca msgassert.h foi levada em consideração, uma vez que o codigo a depender da implementação e caso uma letra seja convertida sem a devida atenção ela pode se tornar um valor numerico, fazendo com que erros em tempo de execução venham a existir, dessa forma, lançar um assert quando o valor passado não era um atributo numérico facilitou o entendimento do programa.

### Conclusões

O trabalho prático é a melhor forma de entender como certas estruturas se comportam e a sua implementação apesar de não ser trivial, acaba se tornando um desafio extremamente produtivo, desenvolvendo habilidades do aluno como programador, mas a cima de tudo, nos fazendo entender como entidades tão abstratas podem ser tão intuitivas com a prática.

Foi feito um Lista encadeada de um objeto do tipo Mensagem, apos isso a lista encadeada foi incorporada atraves de composição a um objeto do tipo usuario que posteriormente foi adicionada a uma outra lista encadeada com objetos do tipo Usuario, esse aninhamento foi fundamental para o entendimento do assunto.

Sumario do que foi entendido com a implementação do trabalho prático.

-Manipulação de string através de funções como find.

- Entendimentedo de como funciona uma lista encadeada.

- Como procurar, exibir e excluir elementos em uma Lista encadeada.

- A Diferença da utilização da alocação dinamica para a alocação estatica de forma muito mais acentuada.

- A importancia real de como é prejuducial vazamentos de memoria.

- A eficiencia da utilização de uma lista feita por alicação dinamica.

### bibliografia

Disponível em: <https://cplusplus.com/reference/string/string/npos/>. Acesso em: 19 out. 2022.

Programar em C++/Manipulando strings - Wikilivros. Disponível em: <https://pt.wikibooks.org/wiki/Programar\_em\_C%2B%2B/Manipulando\_strings>.

Disponível em: <https://cplusplus.com/reference/string/string/substr/>.

Listas encadeadas. Disponível em: <https://www.ime.usp.br/~pf/algoritmos/aulas/lista.html>.

KHINTIBIDZE, L. Como determinar se uma string é um número em C++. Disponível em: <https://www.delftstack.com/pt/howto/cpp/how-to-determine-if-a-string-is-number-cpp/>. Acesso em: 19 out. 2022.

Curso de C. Disponível em: <https://www.pucsp.br/~so-comp/cursoc/aulas/cb70.html>. Acesso em: 19 out. 2022.

Disponível em: <https://pt.stackoverflow.com/questions/88064/retirar-elemento-de-uma-lista-encadeada>. Acesso em: 19 out. 2022.

std::find in C++. Disponível em: <https://www.geeksforgeeks.org/std-find-in-cpp/>.

Linked List | Set 3 (Deleting a node). Disponível em: <https://www.geeksforgeeks.org/linked-list-set-3-deleting-node/>.

GEEKSFORGEEKS. Linked List Data Structure - GeeksforGeeks. Disponível em: <https://www.geeksforgeeks.org/data-structures/linked-list/>.

### instruções

Instruções para compilação e execução: Esta seção deve ser colocada em um apêndice ao fim

do documento e em uma página exclusiva (não divide página com outras seções).

Ao compilar o arquivo é necessário a utilização do MAKEFILE, ele será responsável por gerar todos os arquivos binarios que serão necessários para a compilação do programa.

Ao utilizar o comando make no terminal, será gerado todos os arquivos binários na pasta obj, após isso o arquivo run.out será gerado na basta bin. O programa será executado quando passado um arquivo de texto com todas as opções do servidor de emails, exibindo ao final da execução no terminal do windows ou linux. Foi utilizado o ambiente Linux para todo o desenvolvimento do programa.

Quanto a saida: por padrão todas as opçoes estão sendo exibidas no terminal, por conta disso, caso aja a necessidade de gerar um arquivo para comparação, é recomendado usar o comenado ./bin/run.out ./tests/input.txt > ./tests/saida.txt para gerar o arquivo de texto para futura comparação. Os arquivos extras disponibilizados no moodle se encontram na pasta tests em uma subpasta chamada inputs, assim como as saidas que se encontram na subpasta outputs.