Programação Imperativa

Instituto Politécnico do Cávado e do Ave
Escola Técnica Superior Profissional
CTeSP – Tecnologia e Inovação Informática
Ano Letivo: 2021/2022

Relatório

Trabalho Prático

Diogo Manuel Marques – 23000

Joana Freitas Pimenta – 22999

João Gabriel Teixeira - 23013

Ludgero Miguel Simões – 23135

1. Tema

O tema proposto consiste de um sistema IoT inteligente que realiza a gestão da recolha de resíduos de reciclagem de um município, pretendendo monitorizar em tempo real a localização, o conteúdo e a capacidade de ecopontos, de maneira a permitir proporcionar rotas otimizadas para os veículos de recolha de resíduos.

2. Funcionalidades propostas

Foi-nos proposto desenvolver um programa em que ecopontos, que tem atribuído um tipo específico (papelão, vidrão e óleão) e uma capacidade, localizados através de coordenadas num plano bidimensional (1000*1000), sendo capaz de existir tanto singularmente como múltiplos, podendo ser de tipos diferentes ou iguais.
A um veículo passar e recolher o conteúdo do ecoponto o mesmo fica vazio, e o programa deve permitir ao utilizador adicionar, editar e remover ecopontos, acrescentar resíduos ao ecoponto e realizar pesquisas de maneira a verificar o estado da rede de recolha de lixos.
O programa deverá também manter e utilizar dados armazenados em um ficheiro e permitir o cálculo da distância percorrida pelos veículos.

3. Estrutura analítica do projeto

Estruturamos o projeto em um menu principal, que se subdivide em cada funcionalidade do programa. As funcionalidades são então: um menu para adicionar, um menu para editar, remover, um menu para procuras, cálculo da rota, um menu de listagem, limpar o ecoponto, realizar a média de resíduo, carregar os registos e guardar os registos.
Nós organizamos o projeto em 4 ficheiros, pois achamos mais limpo e objetivo, facilitando a compreensão, e dividimos então o programa no ficheiro main.c, que chama a função principal, no ficheiro structs.h, que contém as structs a ser importadas, no ficheiro functions.c, que contém todas as funções, e functions.h, que declara todas as funções. Utilizamos as bibliotecas <stdio.h>, <stdlib.h>, <locale.h>, <malloc.h>, <stdbool.h>, <limits.h> e <math.h>.

4. Funcionalidades desenvolvidas

Ao correr o programa, procede-se ao aumento do número máximo de registos, e verificando-se que não ocorreu nenhum erro com a incrementação chama-se então o menu principal. Caso ocorra algum erro o programa informa o utilizador e pede ao mesmo para tentar mais tarde, terminando.

Através do menu principal acedemos a todas as funcionalidades do programa.

A primeira opção do menu é um submenu que nos permite adicionar ecopontos ou adicionar resíduos.
A função adicionarEco() é a função responsável por adicionar Ecopontos ao plano. Como é uma função void ela não retorna nenhum valor e também não contém parâmetros. A função pede ao utilizador para introduzir o tipo de ecoponto e a sua localização nas coordenadas x e y, e ao se verificar que os dados introduzidos são válidos, o ecoponto é então adicionado. Caso ocorra algum erro o utilizador é alertado.
A função adicionarResiduos() é a função responsável por adicionar Resíduos a um ecoponto. Também é uma função void, portanto ela não retorna nenhum valor, e não contém parâmetros. Para adicionar resíduos a função pede ao utilizador o ID do ecoponto no qual pretende ser feita a inserção e a quantidade que deseja inserir. Sendo os dados que o utilizador introduz válidos, serão adicionados os resíduos ao ecoponto. Caso contrario, o utilizador é alertado.

A opção seguinte no menu apresenta a funcionalidade de edição, que começa por realizar uma verificação de que existem registos para poder então chamar a função editarEco().
A função editarEco() é uma função void, sem parâmetros, que solicita ao utilizador o ID do ecoponto, e sendo este válido, apresenta um menu para o utilizador selecionar se pretende trocar a localização/posição do ecoponto ou se pretende editar o tipo do ecoponto.
Ao selecionar alterar a localização/posição, chama a função void editarCoordenadas(int posicaoArray) que recebendo como parâmetro a posição atual do Ecoponto no array, apresenta um menu que dá ao utilizador a escolha de trocar a coordenada de x ou de y. Sendo verificado que a coordenada é válida efetua-se, portanto, a edição e a quantidade atual de resíduos do ecoponto fica a 0, pois o mesmo é esvaziado para ser reposicionado.
Ao selecionar editar o tipo do ecoponto, chama a função void editarEcoTipo(int posicaoArray), que recebendo como parâmetro a posição atual do Ecoponto no array, pergunta ao utilizador o novo tipo que pretende atribuir ao ecoponto (1 - papelão, 2 - vidrão, 3 - oleão), verificando se é de facto um tipo válido. Sendo verificado que o tipo é válido, o mesmo é alterado e a quantidade atual de resíduos do ecoponto fica a 0, pois o mesmo é esvaziado para poder receber um resíduo diferente.

A terceira opção permite remover um ecoponto.
É realizada uma verificação para confirmar que existem registos para poder então chamar a função void removerEco(), que primeiro pergunta ao utilizador o ID do ecoponto que deseja apagar e verifica se o mesmo é existente. Se existir passa para um array temporário todos os ecopontos menos aquele que o utilizador deseja apagar, realizando uma limpeza do array principal e atribuindo-lhe os dados do array temporario, que no fim do processo é também limpo.

A quarta opção é um submenu que permite fazer procuras nos registos.
É realizada uma verificação para confirmar que existem registos e então chama a função void menuProcurar(), que mostra ao utilizador um menu onde o mesmo pode selecionar procurar ecopontos por tipo ou procurar ecopontos cheios.
Caso selecionem a primeira opção irá ser pedido ao utilizador que introduza o tipo pelo qual deseja procurar, sendo verificado como um tipo válido o mesmo é passado como parâmetro da função void listartEcopontosTipo(int tipo), que efetua uma procura em todos os registos por tipos iguais ao tipo fornecido, armazena-os num array temporario, e então mostra o ID do ecoponto, as coordenadas x e y, e capacidade atual do ecoponto dos ecopontos encontrados, esvaziando o array temporario no fim.

Sendo selecionada a segunda opção é chamada a função void listarEcopontosCheios(), que percorre o array todo para encontrar ecopontos em que a capacidade atual seja maior ou igual à quantidade máxima dos ecopontos, armazenando-os num array temporario e listando o ID do ecoponto, as coordenadas x e y, e o tipo do ecoponto, por fim limpando o array temporario.

A quinta opção efetua cálculos para fornecer ao utilizador a rota de recolha mais otimizada.
Verifica-se se há registos, e no caso de existirem chama a função void calcRota(). Esta função é responsável por calcular a rota mais eficiente, é perguntado o ponto de partida do utilizador e a percentagem de capacidade dos ecopontos que ele pretende recolher, então a função percorre todos os ecopontos e verifica se existem ecopontos que estão com ocupação da capacidade superior ou igual a percentagem fornecida, sendo estes adicionados ao array de coordenadas a percorrer e utilizando a lógica do algoritmo do vizinho mais próximo para definir a ordem de passagem, definindo assim a rota que é salva num arquivo chamado "rota.txt". Caso ocorra algum erro a salvar no arquivo é possível mostrar no ecrã a rota.

A sexta opção é um menu de listagem dos ecopontos.
É realizada uma verificação para confirmar que existem registos e então chama a função void menuListar(), que mostra ao utilizador um menu onde o mesmo pode selecionar listar ecopontos por tipo, listar ecopontos cheios ou listar todos os ecopontos. Esta opção é similar à opção 4, sendo que a segunda e terceira opção do menu de listagem utiliza as mesmas funções que o menu de procuras. A primeira opção utiliza a função listarEcopontos(), que lista todos os ecopontos registados.

A sétima opção consiste na limpeza dos resíduos de um ecoponto.
Caso existam registos, a função void limparEco() é chamada. Ela começa por pedir ao utilizador para inserir o ID do ecoponto que pretende esvaziar e verifica se é um ID válido. Após a verificação chama a função int procuraEcoponto(int inicio, int fim, int idProcura), que é uma procura binária feita recursivamente e retorna o índice do array onde estão armazenados os dados do ecoponto selecionado. Obtendo então a posição no array do ecoponto, verifica se a mesma é válida, e limpa por fim o ecoponto. Caso o ecoponto já esteja vazio, o utilizador será avisado.

A oitava opção permite ao utilizador ver a média de resíduos.
Verifica-se se existem registos, e então chama-se a função void calcMediaResiduos(), que pergunta ao utilizador o tipo de resíduos do qual deseja calcular a média, sendo este válido, faz um loop que percorre todos os ecopontos, os que forem do tipo que o utilizador escolheu são adicionados a soma e é incrementado o contador. Procede-se então ao cálculo da média (soma / contador), e apresentamos o resultado.

A nona opção carrega registos do ficheiro.
Verifica se existe algum registo, caso exista pergunta ao utilizador se quer mesmo carregar os registos do arquivo, já que, ao carregar os registos, o array principal é limpo para receber os registos dos ecopontos que se encontram no arquivo, perdendo os registos na memória. Se não houver registos ou se o utilizador consentir, é atribuída à variável respCarregarRegistos o retorno da função int carregarRegistos(). Esta função é responsável por carregar os ecopontos do "ecopontos.dat" para a memória. A função retorna 1 caso seja executada com sucesso, 0 se acontecer um erro ao abrir o arquivo, -1 caso haja um erro ao carregar os dados para o arquivo e –2 se houver erro ao alocar a memória para salvar os registos. O utilizador é alertado dos erros.

A decima opção salva registos no ficheiro.

Verifica se existe algum registo, se não existir apresenta uma mensagem de erro. Caso exista, verifica se o utilizador quer mesmo guardar os registos, pois ao guardar, os registos que se encontram no arquivo irão ser apagados. Se o utilizador consentir, é atribuida à variável respGuardarRegistos o retorno da função int guardarRegistos(Ecoponto ecopontoToSave[], int tamanho). Esta função é responsável por receber um array do tipo Ecoponto e o tamanho do mesmo e salvar os ecopontos no arquivo chamado "ecopontos.dat" (Ficheiro Binário). A função retorna 1 caso seja executada com sucesso, 0 se acontecer um erro ao abrir o arquivo e -1 caso haja um erro ao guardar os dados no arquivo. O utilizador é alertado dos erros.

5. Conclusão

De maneira geral, atingimos todos os objetivos propostos para este projeto e ultrapassamos as dificuldades que surgiram.
Algumas dificuldades que enfrentamos foram: problemas a ler tipo char, dificuldade em implementar a matriz com ponteiros, precisamos de criar o nosso "scanf" para evitar que quando introduzíssemos uma letra ele não houvesse bugs, tivemos de testar muito para otimizar e encontrar o melhor script de calcular o vizinho mais próximo e tivemos um problema ao salvar e ler dos arquivos, porque no fim nós dávamos free ao ponteiro do arquivo e igualamo-lo a null e isso gerava um crash no programa, o mesmo problema acontecia quando era um ponteiro para char.