Departamento de Engenharia Informática | Licenciatura em Engenharia Informática

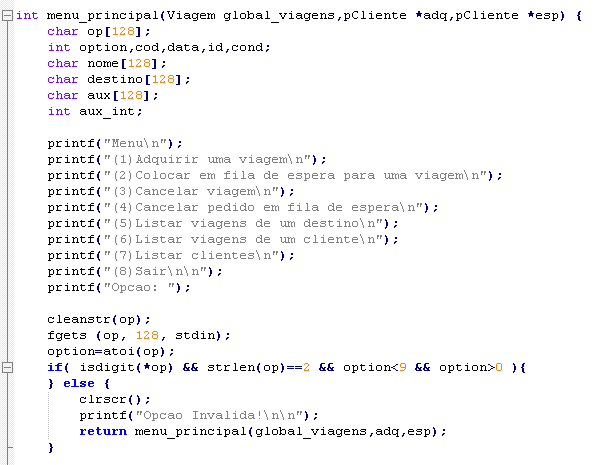
-Manual de programador -

Príncipios de programação procedimental

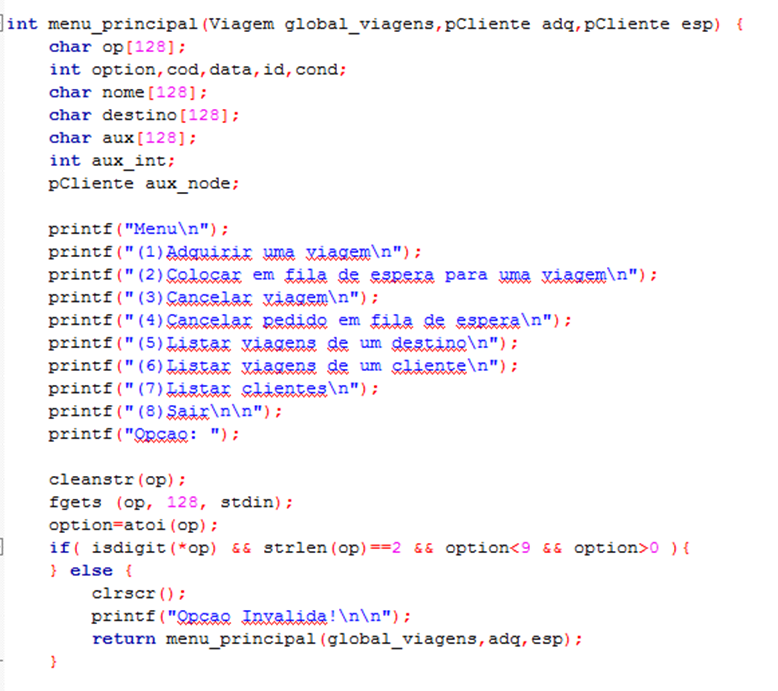
André Loras Leite nº 2015250489 TP2

Gonçalo Oliveira Amaral nº 2015249122 TP2

**Função Menu\_principal:**



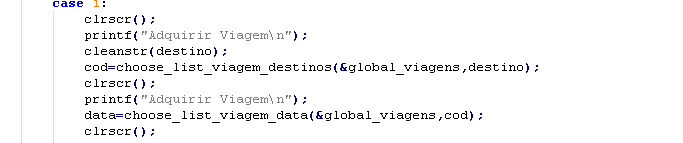
Esta função tem como principal finalidade dispor ao utilizador uma série de opções, num pequeno e simples menu que dependendo da opção inserida irá permitir manipular de forma fácil e com alto nível de abstração, toda a aplicação. Esta mesma função está protegida caso o elemento introduzido seja um dígito fora do limite [1,8] ou qualquer outro tipo de caracter. É aplicado pela condicionada da forma **if -> else** **-> return**:



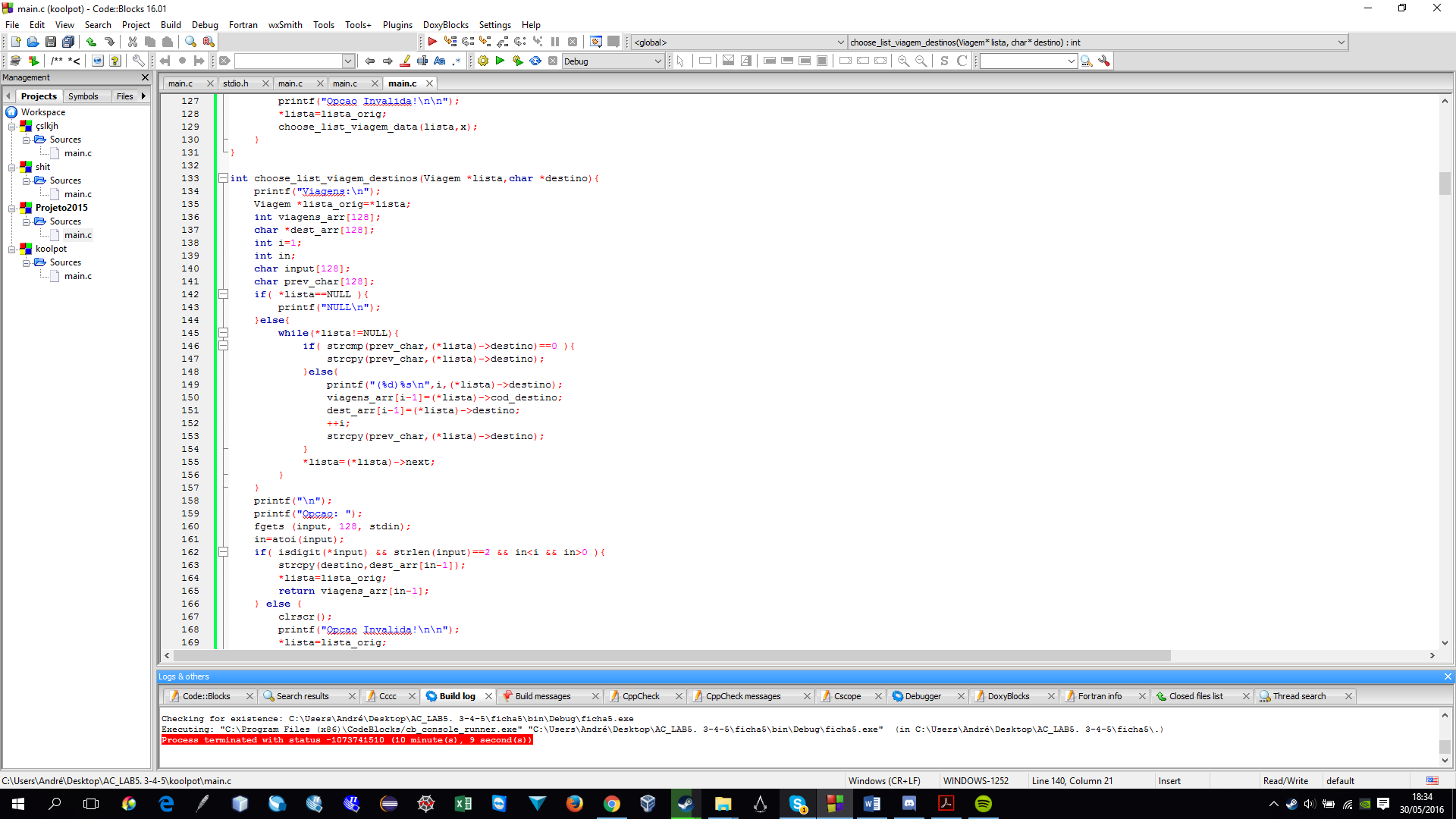
As opções são controladas pela instrução de escolha múltipla **switch** **-> case.** Cada valor está associado a um **case**, e dependendo do valor do parâmetro do **switch**, é executado o respetivo **case.** Vejamos então a implementação desta instrução para a aplicação:



**CASE 1:**

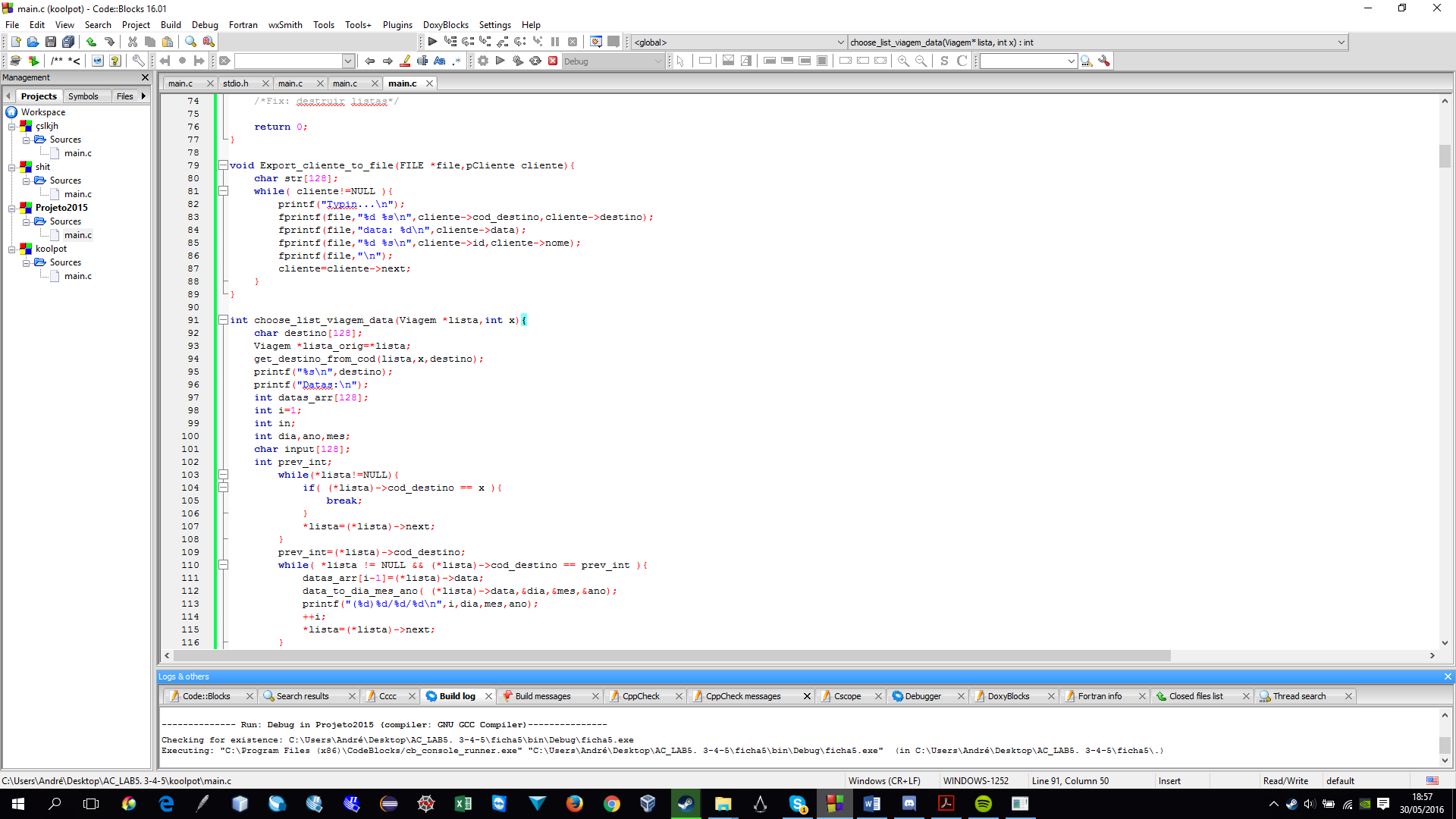


Este caso é direcionado para o inteiro (1) uma vez que corresponde á primeira opção (Adquirir um viagem). Caso o inteiro 1 seja o selecionado esta **case** entra em vigor e começa por fazer um print do nome da opção, por conseguinte é chamada a função **choose\_list\_viagem\_destinos:**



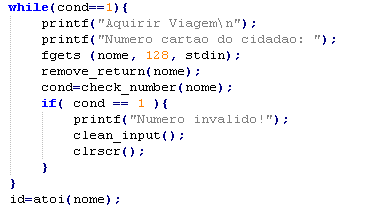
Esta função cria um pequeno menu com todos os destinos. Para além dispor também permite selecionar um destino sendo o valor devolvido pela função o código da viagem.

Em seguida é novamente chamada uma função para complementar a escolha e aplicar as devidas mudanças e manipulações á aplicação. Desta vez chamamos uma função denominada por **choose\_list\_viagem\_data**:

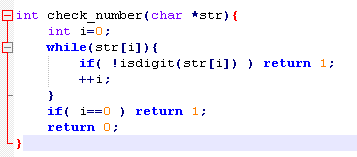


Semelhante à função anterior, esta função dispõem as datas para o destino selecionado, devolvendo a data escolhida.

Uma vez feita a distribuição de dados pelas funções acima referidas e explanadas, podemos passar ao registo do passageiro dada a viagem escolhida, com auxílio das funções **choose\_list\_viagem\_destinos** e **choose\_list\_viagem\_data**. Assim, na condição ***case 1*** e como em todas as outras será possível verificarqueé primeiramente sugerida a introdução do número de cartão de cidadão e isto para evitar que nos registos seguintes seja necessária a introdução repetitiva do nome. Desta forma basta fazer um registo de cartão de cidadão -> nome, que irá ficar registada na base de dados, se o cliente tiver adquirido uma viagem ou colocado uma viagem em fila de espera, relembrando ao sistema que este número de cidadão já foi registado, descartando a nova introdução do nome do cliente.

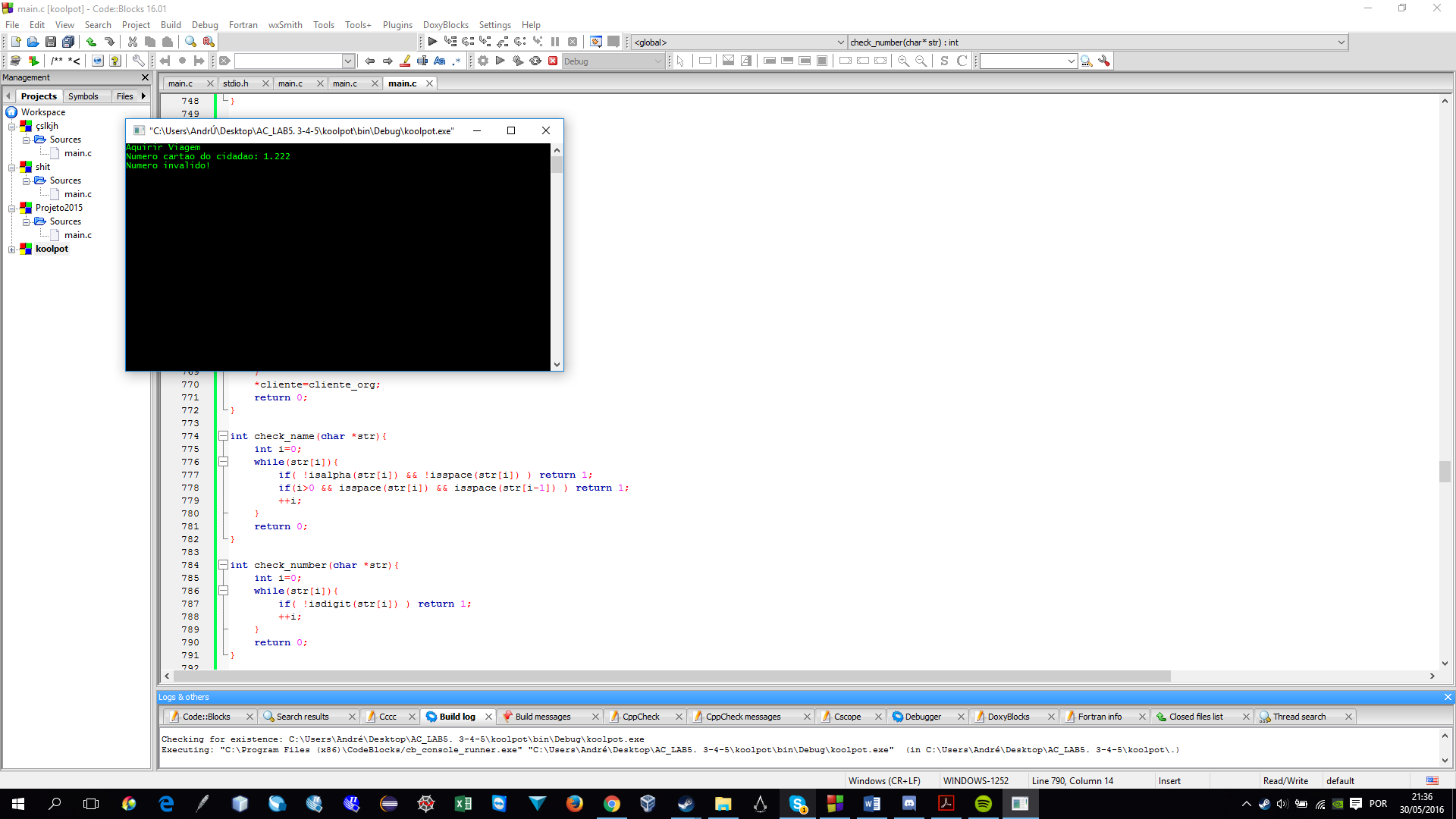


Neste **while loop** temos a introdução do número de cartão do cidadão e a sua respetiva verificação. Começando pela receção de uma string com o respetivo input, de seguida é removido o ‘\n’ recorrendo à função **remove\_return.**

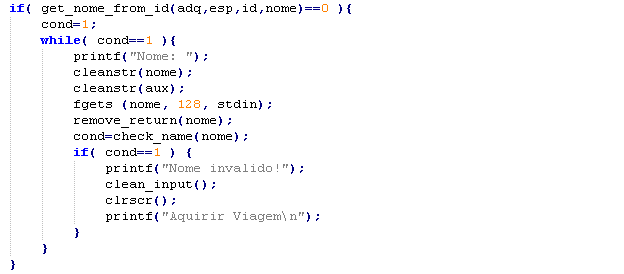


Utilizando a função **check\_number,** verificamos se o input é um número.

Vejamos um exemplo real com a introdução de um número decimal no espaço para o cartão de cidadão:



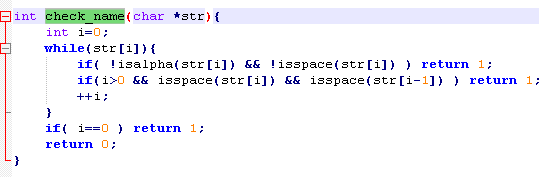
De seguida teremos outro **while loop** onde é feito algo muito semelhante mas com uma restrição adicional, **if clause**  com a função **get\_nome\_from\_id**. Desta vez, é utilizada a função **check\_name** para verificar a string introduzida como parâmetro.



O objetivo principal da função **get\_nome\_from\_id** é verificar se o cartão do cidadão introduzido existe na lista de viagens adquiridas ou na lista de viagens em fila de espera.

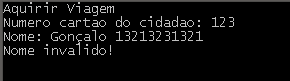


Tem como valor de retorno um inteiro, que é **1** se existir no sistema e **0** caso contrário. Se o número for encontrado, o nome associado ao mesmo é atribuído ao parâmetro nome.



Utilizando a função **check\_name,** verificamos se o input é um nome válido.

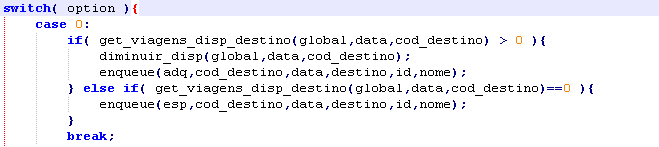
Vejamos um exemplo real com a introdução de um nome seguido de dígitos, no espaço para o nome:

****

Para finalizar este Case, temos ainda a função **insere\_last\_adq:**



Tentamos criar esta função da maneira mais dinâmica possível. O objetivo desta função será inserir no final de uma lista um nó criado através do conjunto de elementos inseridos como parâmetros. Esta função poderá ser cumprir o seu objetivo quer para a lista de viagens adquiridas quer para a lista das viagens em lista de espera através da **option** devendo ser **0** se pretendermos inserir no final da lista de viagens adquiridas e **1** se pretendermos inserir no final da lista de viagens em fila de espera. Para a **option** 0, utilizada no **case 1**, a função para poderá não inserir na lista de viagens adquiridas.



A função recorre irá inserir na lista de viagens adquiridas se existirem lugares disponíveis. Para verificar isto recorremos à função **get\_viagens\_disp\_destino**.



Esta função irá percorrer a lista de viagens e obter os lugares disponíveis para a data e destino introduzidos, usando este valor como valor de retorno.

Se este valor for superior a **0** a função irá inserir na lista de viagens adquiridas, diminuindo a os lugares disponíveis em 1 para a data e destino selecionados.

Para efetuar esta diminuição recorremos à função **diminuir\_disp**



Para efetuar a inserção no final da lista é usada a função **enqueue**



Se o valor da disponibilidade for **0** a viagem será inserida no final da lista de viagens em fila de espera, recorrendo à **enqueue**.

**CASE 2:**

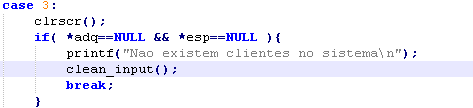
Tal como o anterior caso, as funções utilizadas são três, **choose\_list\_viagem\_destinos**, **choose\_list\_viagem\_data**, **remove\_return** , **check\_name**, **check\_number** , **get\_nome\_from\_id** e **insert\_last\_adq**.

C:\Users\Gonçalo\AppData\Local\Microsoft\Windows\INetCache\Content.Word\Capture.png

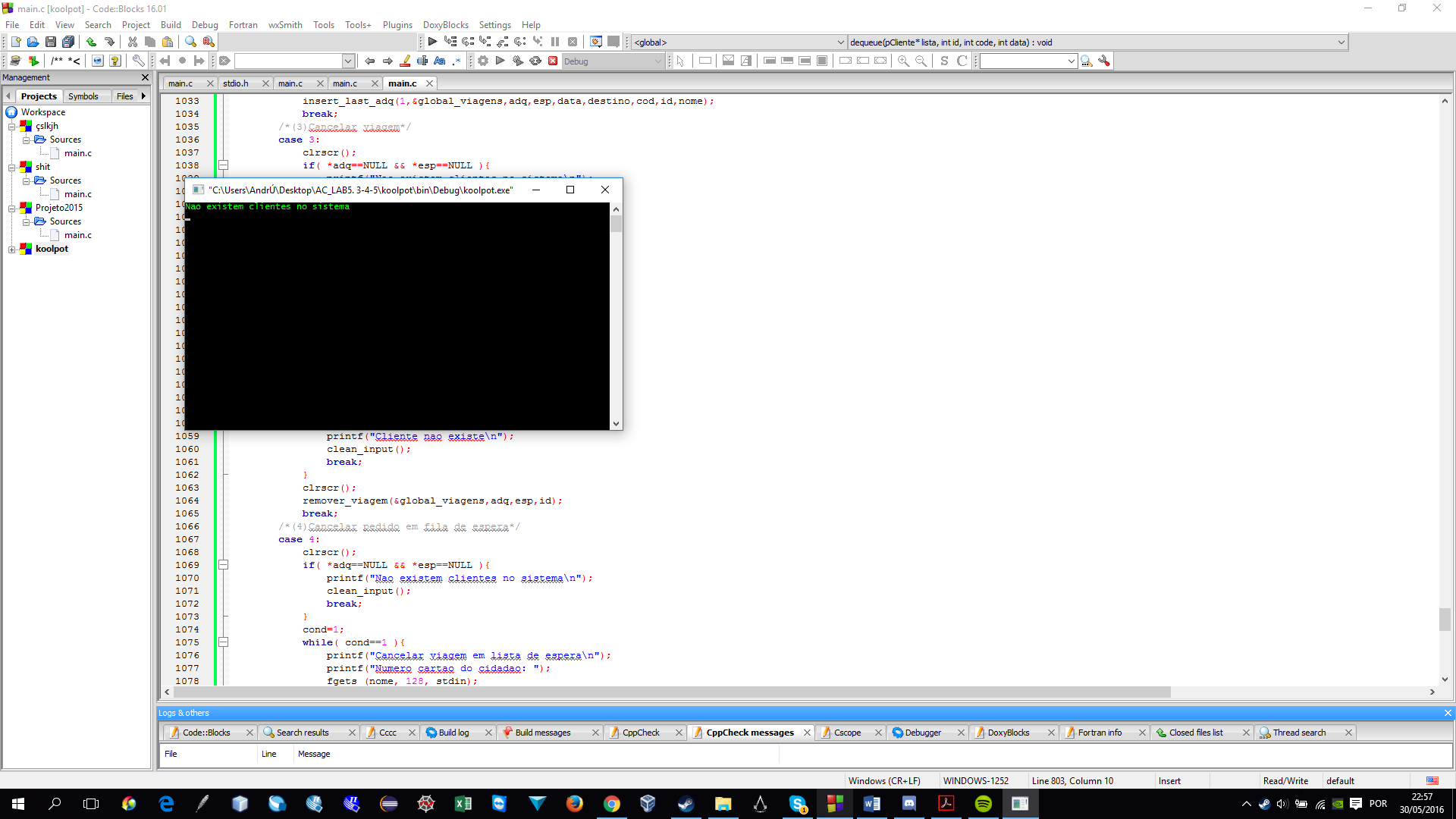
Para perceber a sua funcionalidade basta verificar a explicação da **case 1,** pelo que não existem variâncias de funções, muda somente o objetivo para o qual a informação é dirigida, recorrendo à função **insert\_last\_adq** com o parâmetro **option** a 1. Como foi referido anteriormente, e aqui é apresentado quão dinâmica a função **insert\_last\_adq** pode ser.

**CASE 3:**

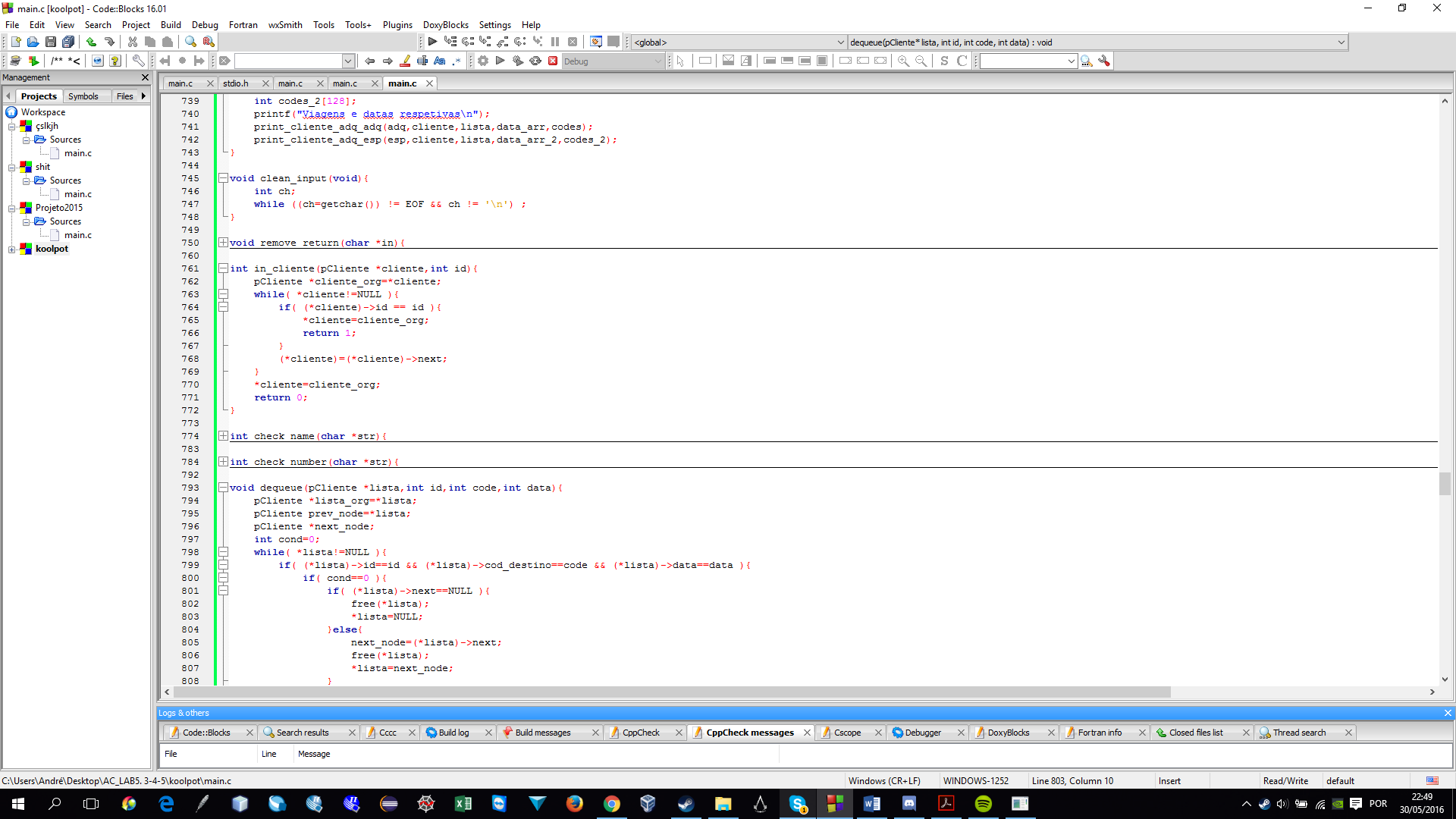
Esta case é direcionada á eliminação de um registo de uma viagem, ou seja, cancelar uma viagem. Para que seja possível fazer o cancelamento é necessário ter a certeza que existem clientes e uma viagem. Assim sendo, temos :



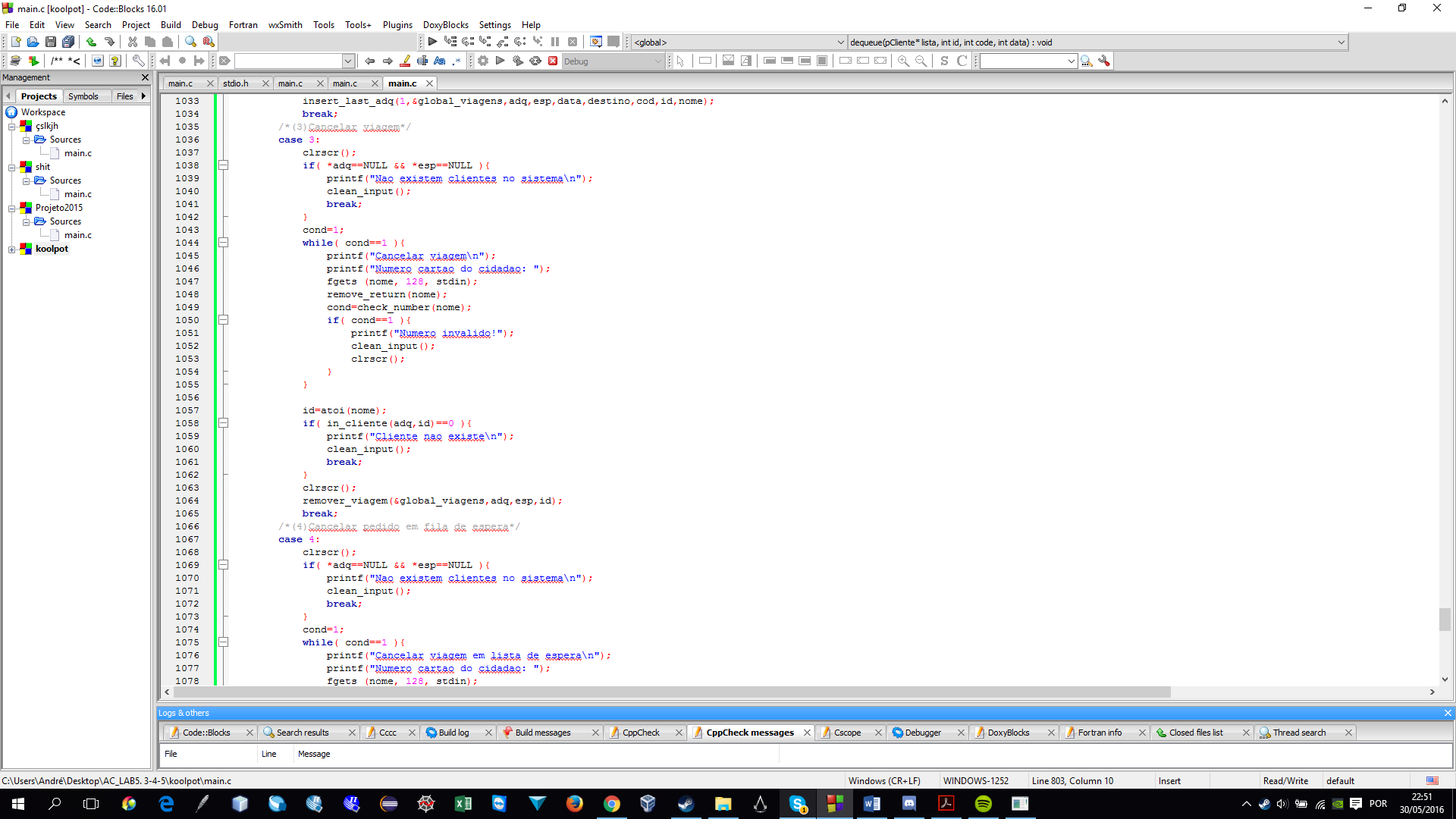
Isto refere-se ao facto de não existir clientes no sistema. Se existirem clientes é necessário fazer um rastreio às viagens que estão registadas pelo cliente. Assim, temos que:



Esta solução é complementada pela função **in\_cliente**, que faz a verificação da existência de um cliente numa lista especifica dado o seu número de cartão de cidadão. Se for encontrado um cliente referente ao número introduzido é feito um **return** de **1**, caso contrário **0**.



Caso o número de cartão de cidadão introduzido não seja referente a nenhuma cliente é imprimida uma notificação de erro pela aplicação “Cliente não existe”:



A parte fulcral desta opção é dada pela função **remover\_viagem:**



Esta função tem como intuito eliminar o registo de uma viagem dado um cliente. Esta função é complementada com as funções **print\_cliente\_adq\_adq**, **dequeue\_and\_save**, **enqueue**(já referida), **aumentar\_disp** e **dequeue**.

**print\_cliente\_adq\_adq**



Esta função dispõe no ecrã todas as viagens adquiridas pelo cliente introduzido, dispondo em primeiro as mais antigas e devolvendo como valor de retorno a quantidade de opções disponíveis. Para fazer esta ordenação recorremos à função **bubblesort\_viagens**, **data\_invert** e à função **data\_fix**

**data\_invert**



Esta função recorre a outra função(**data\_to\_dia\_mes\_ano**), que separa a data em dia, mês e ano e inverte a data para o formato ano mês dia para que a função seguinte possa fazer a organização de forma coerente.

**bubblesort\_viagens**



Esta função organiza o vetor de inteiros **vect** por ordem crescente, aplicando as mesmas alterações ao vetor **codes**.

**data\_fix**



Esta função repõe as mudanças feitas pela função **data\_invert**

**dequeue**

C:\Users\Gonçalo\AppData\Local\Microsoft\Windows\INetCache\Content.Word\Capture.png

À semelhança da função **enqueue**, **dequeue¸**faz o inverso e remove o primeiro nó da lista introduzida.

**dequeue\_and\_save**

**C:\Users\Gonçalo\AppData\Local\Microsoft\Windows\INetCache\Content.Word\Capture.png**

À semelhança da função **dequeue**, **dequeue\_and\_save¸**faz o inverso e remove o primeiro nó da lista introduzida e guarda a informação do nó nos respetivos argumentos.

**aumentar\_disp**



À semelhança da função **diminuir\_disp**, **aumentar\_disp** aumenta o valor da disponibilidade de da viagem na data e código de destino usados como argumentos da função.

**CASE 4:**

Bem como no **case 3**, neste **case** é necessário fazer ‘’check-ups’’ à existência de clientes neste parâmetro, ou seja, para que seja possível seguir em frente para o cancelamento de listas de espera é prioritário que sejam detetados clientes pois se o ficheiro estiver vazio a aplicação automáticamente irá cancelar o procedimento imprimindo um aviso ‘’Não existem clientes no sistema’’.

Por conseguinte e como em todos os casos, é pedido o número de cartão de cidadão que vai servir para registar o cliente ou para verificar a sua existência.

No final deste **case** é realizada, tal como no caso anterior, a eliminação do registo da lista de espera, recorrendo à função **remover\_espera**.



Esta função tal como a anterior **remover\_viagem**, utiliza funções secundárias ou de controlo para permitir a eliminação das listas de espera em ficheiros. Nesta função é permitida o cancelamento por opção, ou seja, é possível escolher a viagem que se quer cancelar, tal como era previsto nos objetivos da aplicação.

**CASE 5:**

Nesta **case** o objetivo é listar todas as viagens disponíveis de um determinado destino, sendo as mais recentes apresentadas em primeiro. Assim, para este efeito, foram usadas duas funções para aglomerar a informação de forma organizada. As funções são denominadas por **choose\_list\_viagem\_destinos** (anteriormente referina, **case 1**) e **print\_viagens\_mais\_recentes**.

Como já foi explorada a finalidade da função **choose\_list\_viagem\_destinos** passemos à utilidade da função **print\_viagens\_mais\_recentes**.



Esta função utiliza a função **get\_destino\_from\_cod**,  **bubblesort\_inv**, **data\_invert**(referida anteriormente) e **data\_fix**(também já referida) e cria um menu onde selecionamos o destino e é nos apresentado uma lista das datas para esse destino com as mais recentes primeiro.

**get\_destino\_from\_cod**



Esta função permite, mediante a introdução de um código de destino, a atribuição do nome do destino correspondente à string **destino**.

**bubblesort\_inv**



Esta função organiza o vetor **vect** por ordem decrescente.

**CASE 6:**

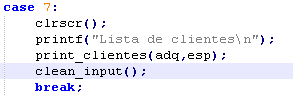
Em termos de acessibilidade é muito semelhante às repetivas cases 1 e 2. O método de reconhecimento e procura é igual e só muda a implementação da função **print\_cliente\_adq.** Esta função é completada por três outras, **print\_cliente\_adq\_adq** (anteriormente referida) e **print\_cliente\_adq\_esp:**

Quanto á função **print\_cliente\_adq\_esp** a sua funcionalidade basea-se em algo muito semelhante a **print\_cliente\_adq\_adq** mas esta aplica-se à lista de viagens em fila de espera.

A função **print\_cliente\_adq** não passa da combinação de **print\_cliente\_adq\_adq** e **print\_cliente\_adq\_esp**

**CASE 7:**

Utilizando a função **print\_clientes** é possível listar na consola todos os clientes uma vez que estes já se tenham registado ao efetuar uma compra ou a colocar uma viagem em fila de espera, de outra forma não será possível imprimir quaisquer clientes uma vez que não foram registados no sistema.



**CASE 8:**

Este case existe meramente para dar ao utilizador a opção livre de terminar a aplicação uma vez que a sua opção resulta num **break** que será definido como a paragem total do ciclo envolvente (**Switch** **case**).

**Função main**

Na função main são feitas a importação dos ficheiros para listas e após mudanças, as respetivas alterações naslistas nos ficheiros.

**Função Export\_cliente\_to\_file**

Esta função tem o objetivo de exportar uma lista do tipo **pCliente** para um ficheiro.

**Funções destroi\_lista\_viagem e destroi\_lista\_cliente:**

Tal como o nome sugere, estas funções destroem as listas no final do programa.

**Funções CreateGlobalListFromFile, createAdquiridasListFromFile, createEsperaListFromFile**

Tem como função importar os dados dos ficheiros para as listas correspondentes

**Funções Secundárias**

Funções **getlinename** e **getlinecode:**



Sabendo que a string **line** tem um determinada estrutura, copia o nome dessa **line** para a string **aux**. Se não foi possivel executar a cópia do nome é retornado **0.**



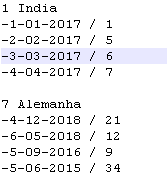
Sabendo que a string **line** tem um determinada estrutura, devolve o numero como valor de retorno.

**Funções print\_list\_viagem e print\_list\_clientes:**

Estas funções são utilizadas para mero “debug” mostrando de forma organizada a lista introduzida.

**Estrutura de ficheiros**

**viagens-data.txt**



**Linha Cabeçalho:** (código\_destino) (espaço)(destino)(“\n”)

**Linha datas e lotação:** (“-”)(dia)(“-”)(mes)(“-”)(ano)(espaço)(“/”)(espaço)(lotação)(“\n”)

**Linha de separação:** (“\n”)

**viagens-adquiridas.txt e viagens-espera.txt**



**Linha Cabeçalho:** (código\_destino) (espaço)(destino)(“\n”)

**Data:** (data)(“\n”)

**CC e nome:** (CC)(espaço)(nome)(“\n”)

**Linha de separação:** (“\n”)

**Estrutura de variáveis específicas e justificação da sua implementação**

**Data**

**Exemplo:** 1022015

(dia - 1)(mês - 02)(ano - 2015)

Utilizamos esta estrutura para facilitar a movimentação da data em toda a aplicação em vez de movermos 3 variáveis, dia, mês, ano, apenas movimentamos um formato compactado.

**Código destino:**

**Exemplo:** 1 India

Código de destino: 1

Decidimos associar a um destino um código pois é mais eficiente e fácil comparar dois inteiros e usar inteiros como parâmetros do que uma string. Por exemplo, o simples processo de procurar um nó que corresponda ao destino inseria, necessitaria de vários **strcmp**, e não podemos duvidar que comparar um numero seria muito mais rápido que comparar uma string.