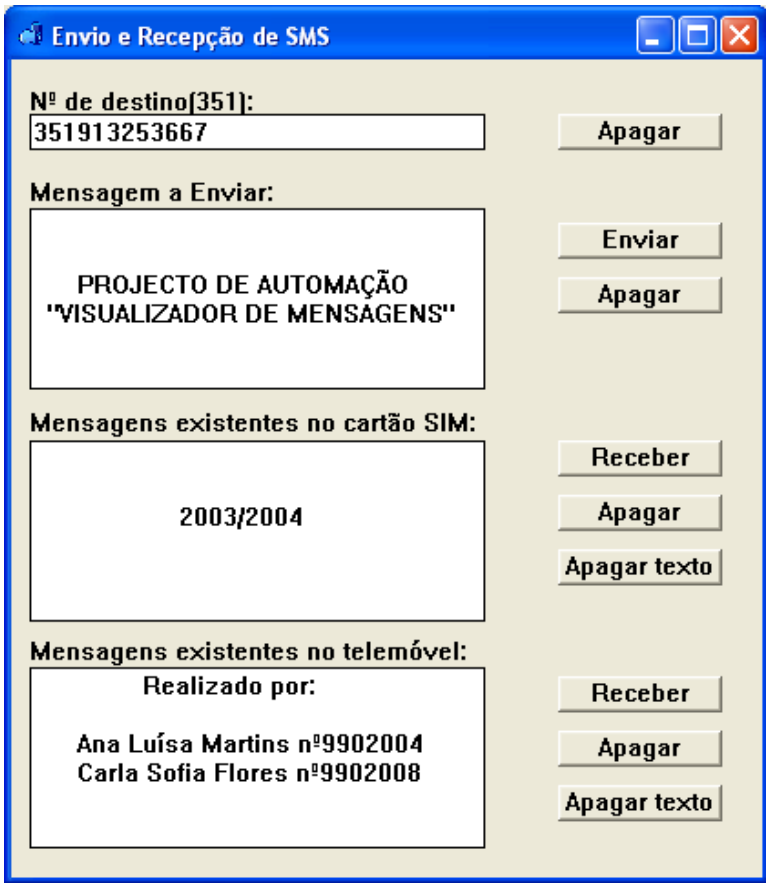
INSTITUTO SUPERIOR DE ENGENHARIA DE COIMBRA



ÍNDICE

1. INTRODUÇÃO

1.1. Objectivo

"Pretende-se com este projecto desenvolver a interface de comunicação série com um telemóvel, de forma a permitir o controlo e monitorização de sistemas à distância através da rede celular GSM. Na aplicação que se pretende implementar é usado um computador pessoal PC para fazer o controlo de todo o sistema."
O software que irá ser utilizado é o Visual C++ para Windows.

1.2. Descrição da Interface entre o TE e o MT

1.2.1. Como enviar e receber mensagens através de um telemóvel (MT- Mobile Termination) e da rede GSM/UMTS a partir de um computador (TE- Terminal Equipment) passando por um adaptador (TA- Terminal adaptor)

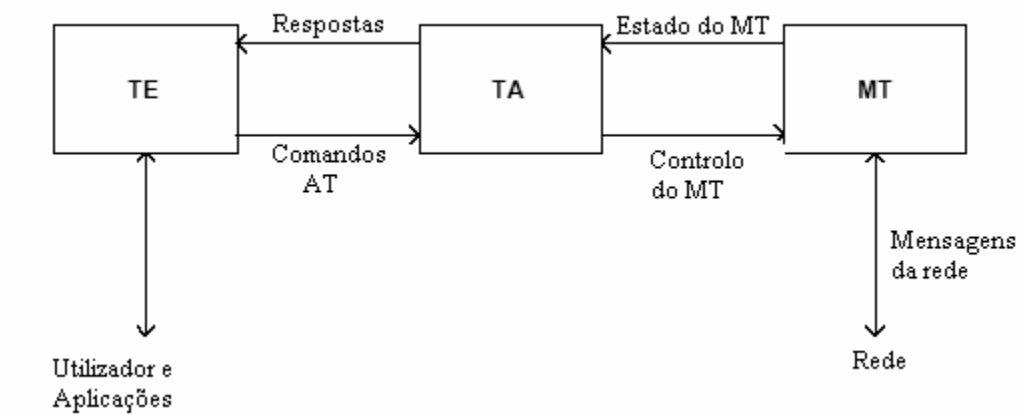


Fig.1-Ligação existente entre um TE e MT

O interface entre o TE e o TA poderá ser feito por cabos, infravermelhos ou com outras ligações de comportamento semelhante, no nosso caso irá ser feita por cabo (RS232).
Para um bom funcionamento os comandos requerem 8 bits de dados, então a ligação TE-TA requer o modo 8 bits/byte.

1.3. Protocolos de interface para o controlo de funções SMS para um telemóvel GMS/UMTS:

1.3.1. Protocolo binário Modo Bloco – este protocolo inclui protecção de erro e é satisfatório para usar onde a ligação pode não estar completamente segura. Será de uso particular onde os aparelhos de controlo remoto são requeridos. A transferência de dados binários codificados do usuário é possível.

1.3.2. Modo Texto– baseado numa interface comandos AT ("Text Mode"). Este modo é satisfatório para terminais não inteligentes e para construção de software de aplicação em estruturas de comando como as definidas em V.25ter (1). Alguns comandos definidos neste protocolo serão também úteis nas implementações do 1º protocolo (Modo Bloco) e do 3º protocolo, por exemplo permitir a indicação de mensagens SMS não enviadas.

1.3.3. Modo PDU – baseado na transferência binária "hex-encoded" de blocos mensagem (Modo PDU). Este é o modo mais robusto e utilizado, uma vez que nem todos os telemóveis possuem o modo texto.

Nestes 3 modos, o terminal é considerado para estar no controlo de transacções SMS.

Na figura 1 estão representados os 3 modos:

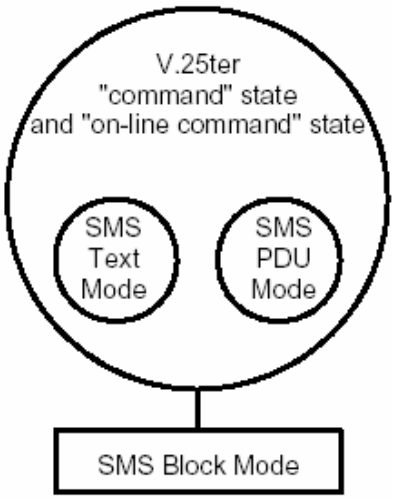


Fig.2 – Modos Bloco, Texto e PDU

O Modo Bloco é um modo independente e quando accionado, o controlo continuará com este modo até os procedimentos de saída do modo serem executados, depois cada controlo é retornado para o estado "command" de V.25ter ou estado "on-line command".

Os modos "Text" e "PDU" são simplesmente ajustes de comandos e constituem estados transitórios que operarão quer no estado "command" de V.25ter ou estado "on-line command" e depois de cada operação o controlo é automaticamente retornado para um destes estados de V.25ter. No estado "command" de V.25ter, o MS está disponível para manobrar chamadas que estão para serem recebidas e enviadas.

(1) v.25ter- protocolo standart desde Julho de 1997, que consiste na ligação e controlo automáticos assíncronos serialmente.

2. DESENVOLVIMENTO

2.1. Modo Bloco SMS

A interface (DTE-DCE) do TE com o MT está associada com a função de adaptação a um terminal (TAF), se esta estiver disponível.
Para entrar neste modo deve-se enviar o comando 'AT+CESP' do TE para o MT .
Na recepção deste comando o MT poderá responder:

- 'OK' (ou 0) se suportar os comandos do modo bloco e entrar no mesmo.
- 'ERROR' se não suportar os comandos do modo bloco e permanecer no modo em que se encontra.
- O software terminal deverá aguardar um pequeno período de tempo (+/- 5s) para as respostas 'OK' (0) ou 'ERROR' (4). Se nenhuma destas for recebida antes do fim deste período de tempo, então o software terminal assumirá que o modo bloco "entrou". O software terminal pode então submeter o seu primeiro comando modo bloco.

Para regressar ao modo 'default' deverá emitir o comando 'END SMS MODE' para o TE, este voltará ao modo ´default' caso uma resposta seja recebida, ou não.

Se uma 'chamada de dados' chega enquanto a interface DTE-DCE está fixada no modo SMS/CBS, então o MT pode independentemente emitir a indicação 'END SMS MODE' e reverter para o modo 'default' em ordem a proceder à 'chamada de dados' através do TAF.

Uma condição BREAK em qualquer direcção da interface DTE/DCE causa a saída do modo bloco SMS/CBS e o regresso ao modo 'default'.

2.1.1. Descrição do Protocolo

O caminho de comunicação entre o MT e TE pela interface DTE/DCE é bastante seguro se usar uma ligação de fio condutor curta. No entanto, para assegurar que a baixa taxa de erro não cause mau funcionamento, o seguinte esquema de protecção de erro é implementado:

Cada mensagem enviada do MT para o TE, ou vice-versa, consiste num bloco de dados (DATA) e num bloco 'check sum' (BCS). Na seguinte descrição as notações DLE, STX, NUL e ETX referem-se às características de controlo tendo os valores hexadecimais 10 02 00 e 03 respectivamente.

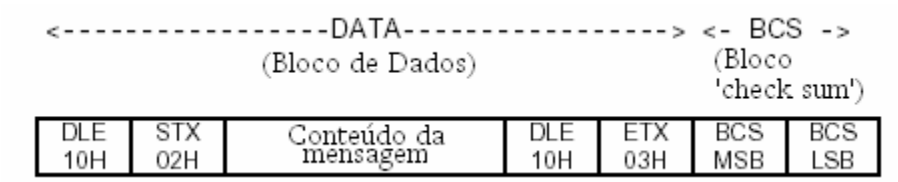


Fig.3 – Formato de mensagens de interface DTE/DCE

O bloco de dados consiste numa sequência de transmissão inicial fixada para 00010000 00000011 (10 02 hex), no conteúdo da mensagem e numa sequência de transmissão final fixada para 00010000 00000011 (10 03 hex). O bit menos significativo de cada octeto é sempre transmitido em 1º lugar.

O bloco 'check sum' é calculado ao transmissor, somando todos os octetos do conteúdo da mensagem (modulo 65536). Cada bit do resultado de 16-bit é depois invertido, e '1' é adicionado à resposta.

Durante a transmissão do conteúdo da mensagem e dos octetos BCS, qualquer ocorrência do valor 10 hex (DLE) deverá resultar num octeto 'stuffing' (de enchimento) adicional de valor 00 hex (NUL) sendo transmitido imediatamente seguindo o octeto contendo 10 hex. Isto para assegurar que as marcas do principio e do fim sejam claramente interpretadas. O receptor deverá remover os octetos 'stuffing' descartando qualquer octeto de valor 00 hex (NUL), o qual segue imediatamente um octeto de valor 10 hex (DLE).

Depois da remoção de qualquer octeto 'stuffing', o receptor pode verificar o BCS, somando todos os octetos no conteúdo da mensagem e o 16-bit BCS modulo 65536. O resultado correcto é 0000 hex. Se qualquer mensagem é recebida com um BCS incorrecto, então a mensagem é descartada. Nenhuma resposta é enviada sobre a interface DTE/DCE.

(Nota – O Modo Bloco foi implementado até aqui no VC++, uma vez os telemóveis por nós utilizados (Nokia 6210 e Siemens M55) não possuem este modo)

O transmissor deverá apenas enviar DLE quando é seguido por STX, NUL ou ETX. Então se o receptor vê um DLE seguido por outra coisa qualquer assumirá que alguns dados foram perdidos e deverá começar a procurar a marca de início. Uma marca de fim inesperada no receptor deverá resultar também na procura de uma marca de início. Uma marca de início deverá sempre ser tratada como o princípio de um novo bloco, independentemente do estado em que o receptor se encontra.

Somente uma transacção comando/resposta deverá ser permitida em qualquer vez de uma identidade receptora ou emissora. Deverá no entanto ser possível para uma transacção comando/resposta de uma identidade, ser iniciada mesmo se uma transacção comando/resposta da outra identidade estiver em curso.

Se é esperada uma resposta imediata para uma mensagem enviada sobre a interface DTE/DCE, então a identidade emissora deverá aguardar 10s. Se nenhuma resposta é recebida neste tempo, então o emissor deverá repetir a mensagem. A mensagem deverá ser repetida no máximo 3 vezes, depois disto o emissor deverá sair do modo SMS/CBS e indicar um erro ao utente.

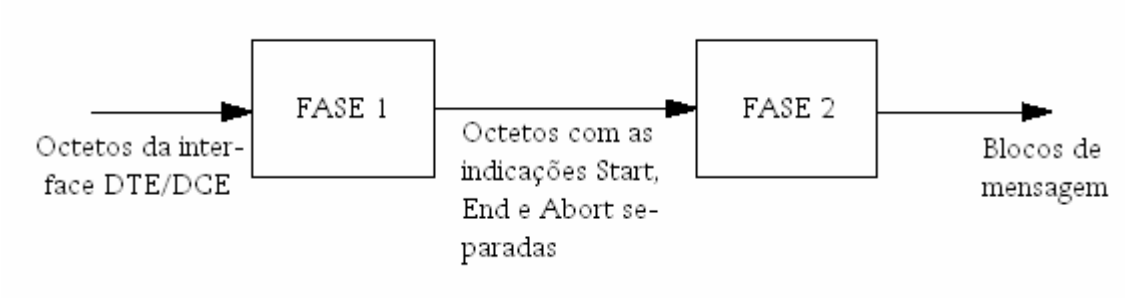
Se uma mensagem não for entendida pela identidade receptora mesmo tendo um BCS correcto, deverá retornar uma mensagem UNABLE TO PROCESS com valor de causa ‘command not understood’. O recebimento de uma mensagem UNABLE TO PROCESS não deverá por si mesmo iniciar a retransmissão, embora esta possa acontecer devido ao mecanismo ‘tempo acabou’ (10s) descrito em cima desde um UNABLE TO PROCESS ser julgado para ser uma resposta inválida. A ‘causa’ pode no entanto ser referida para uma camada mais alta. Uma mensagem UNABLE TO PROCESS não deverá ser enviada como resultado de um incorrecto BCS.

2.1.1.1. Exemplo do diagrama de estados para o bloco receptor

O diagrama de estados seguinte mostra como a componente receptora no nível bloco pode funcionar. Neste exemplo os octetos recebidos são processados em duas fases.

A Fase 1 é uma função de baixo nível, a qual detecta as marcas únicas de início e fim, e remove qualquer octeto ‘stuffing’. Os resultados desta fase são passados para a fase 2, qualquer valor de octeto inesperado depois de DLE será indicado como ‘abort’.

A Fase 2 reúne o conteúdo da mensagem e os octetos BCS, utilizando os octetos provenientes da fase 1 e as indicações ‘start’ e ‘end’. Uma indicação ‘start’ reajustará sempre o processo para a fase 1 de qualquer fase. Uma indicação ‘abort’ deverá sempre causar o retorno à fase 0 onde uma indicação ‘start’ é aguardada. Quando uma indicação ‘end’ é recebida na fase 1, os dois octetos seguintes são verificados como BCS. Se o BCS é correcto, o conteúdo da mensagem é passado para outra fase do receptor.



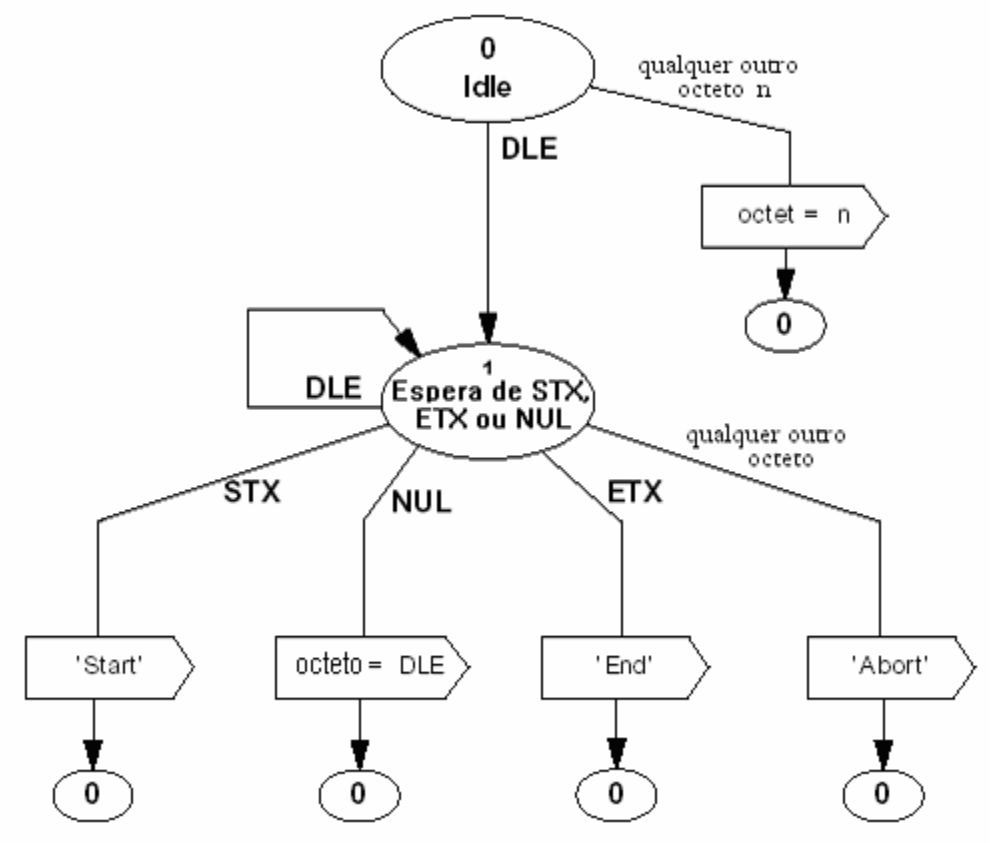


Fig.4 - Transição de estados na fase 1

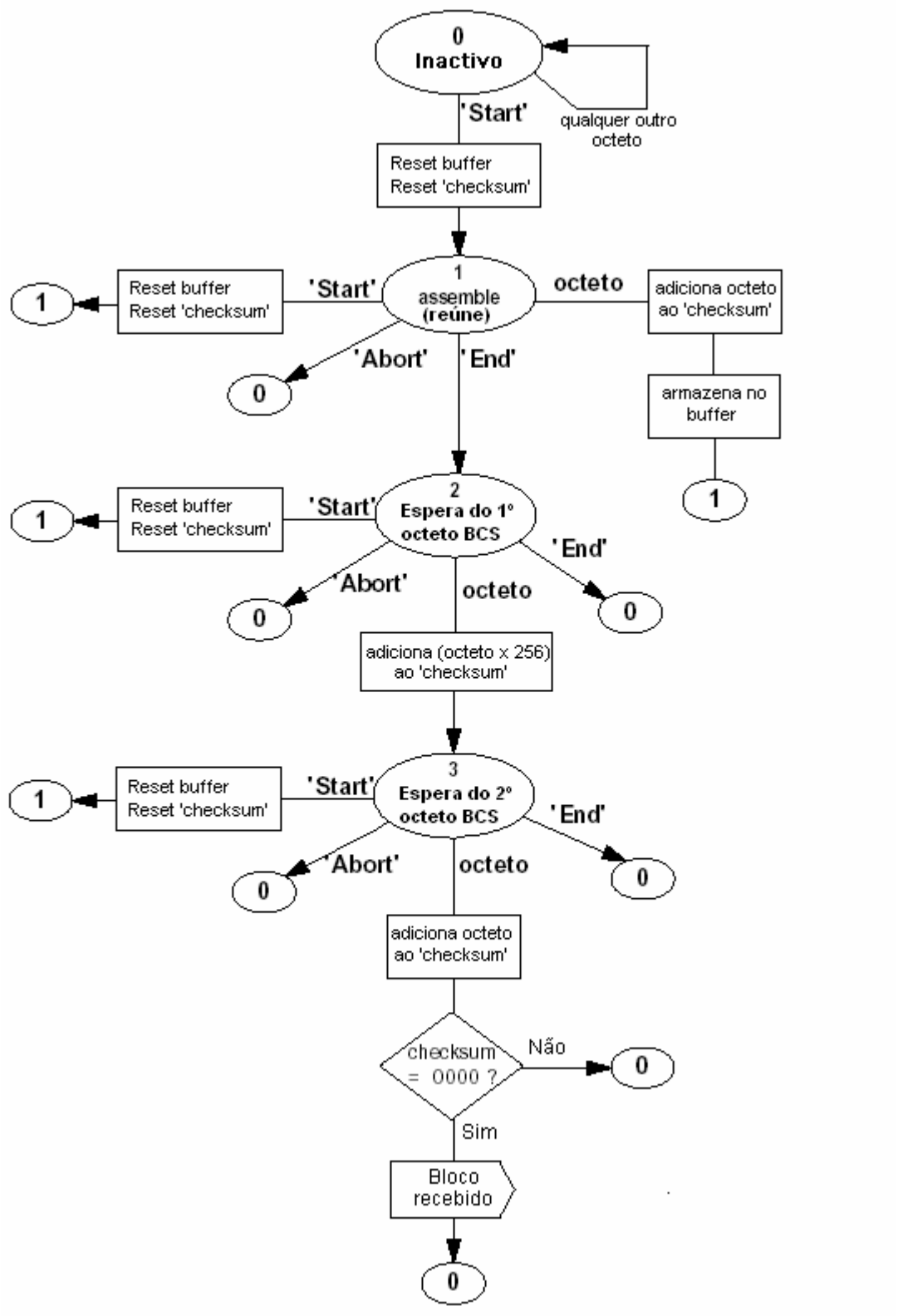


Fig.5 - Transição de estados na fase 2

2.1.1.2. Exemplo de codificação e descodificação de um bloco de dados

Neste ponto é mostrada a codificação de uma mensagem exemplo no transmissor, e os estados de descodificação no receptor, o qual possui as duas fases de processamento do exemplo anterior. Neste exemplo tanto o conteúdo da mensagem como o BCS contêm uma octeto com o valor 10 hex. Então a mensagem como transmitida sobre a interface tem octetos ‘stuffing’ adicionais (00 hex) inseridos depois destes octetos. O receptor primeiro detecta as marcas de início e fim, e remove os octetos ‘stuffing’. Finalmente o BCS é verificado.

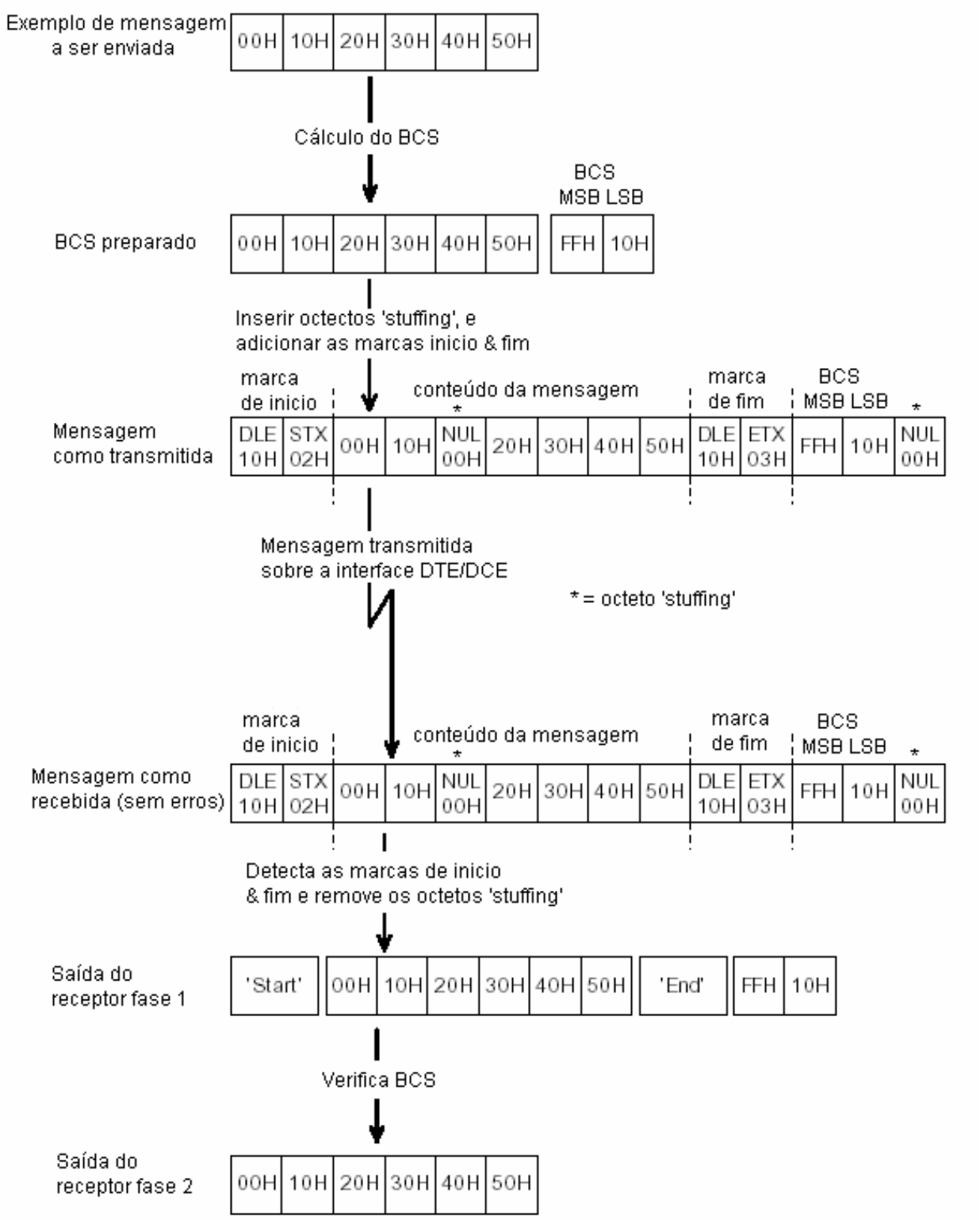


Fig.6 - Exemplo de codificação/descodificação de uma mensagem na interface DTE/DCE

2.1.1.3. Resultado da programação implementada no VC++ :

(Código em anexo, ficheiro 'cod_descod.cpp')

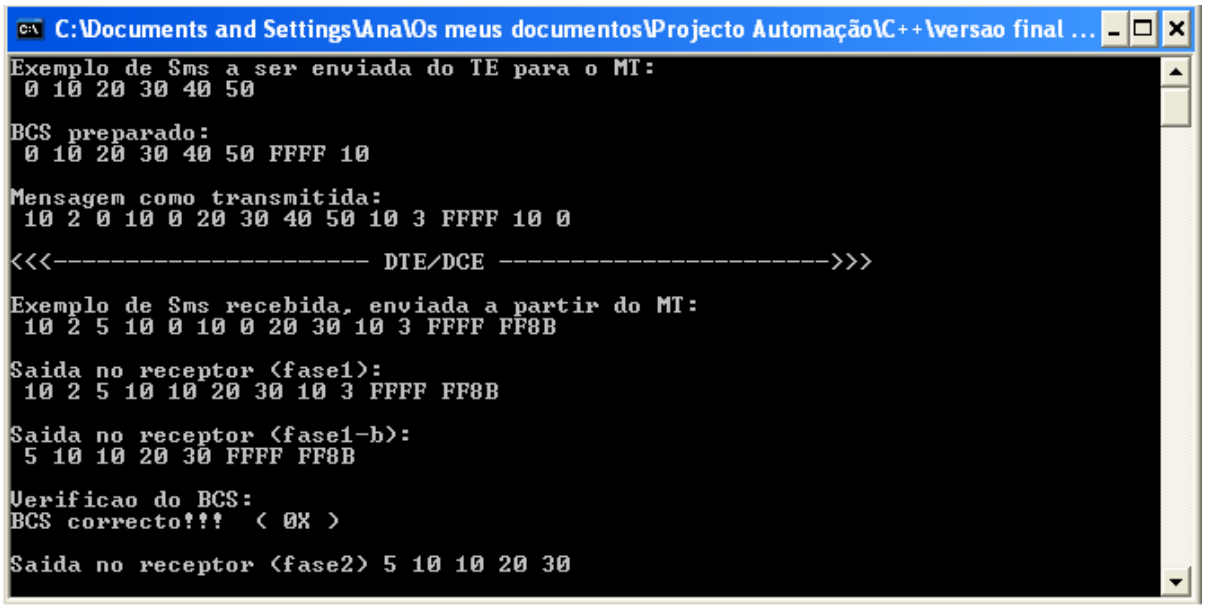


Fig.7 – Resultado no VC++ da implementação do Modo Bloco

2.1.1.4. Pedido de mensagens que já estiveram armazenadas no MT

O TE pode requerer que o MT gere mensagens SMS que já estiveram armazenadas. Pode pedir todas as mensagens, ou uma lista de mensagens e subsequentemente pedir mensagens específicas.

No início da sessão modo SMS/CBS, o MT deverá numerar todas as mensagens continuamente, começando com a mensagem nº1. Estas 'Short Message References' são somente válidas para uma única sessão modo SMS/CBS. Cada mensagem retém a sua 'Short Message Reference' para a duração da sessão modo SMS/CBS. Novas mensagens são normalmente cedidas à mais baixa 'Short Message Reference' previamente não utilizada.

'Short Message Reference '0' significa que não existem mensagens no MT. O valor '0' é usado sob as seguintes condições:

- Quando um comando INSERT SMS é usado para transferir uma SM através de interface "aérea" e não armazene no MT então o MT retornará uma 'Short Message Reference' de 0 na resposta REQUEST TO CONFIRMER e resulta nas indicações INSERT SMS COMPLETE/INSERT SMS FAILURE

- Para Sm's classe 0 que não são armazenadas no MT

- Para Sm's específicas do TE que não são armazenadas no MT

2.1.1.4.1. Comandos emitidos pelo TE:

1. List Request - Esta mensagem é enviada pelo TE para o MT para requerer a lista de mensagens armazenadas no MT.

2. Get Message - Esta mensagem é enviada pelo TE para o MT para requerer a transferência de uma mensagem SMS ou CBS específica armazenada no MT.

3. Get First Message - Esta mensagem é enviada pelo TE para o MT para requerer a transferência da primeira mensagem SMS ou CBS disponível armazenada no MT.

4. Get Next Message - Esta mensagem é enviada pelo TE para o MT para requerer a transferência da próxima mensagem SMS ou CBS disponível armazenada no MT.

5. Transfer Inc SMS - Esta mensagem é enviada pelo TE para o MT para requerer a transferência directa de uma mensagens 'incoming' da interface aérea para o TE.

6. Indicate Inc SMS - Esta mensagem é enviada pelo TE para o MT para requerer que o MT indique quando uma mensagem 'incoming' chega.

7. Transfer Inc CBS - Esta mensagem é enviada pelo TE para o MT para requerer a transferência de toda a radiodifusão de mensagens directamente da interface aérea para a interface DTE/DCE.

8. Insert SMS - Esta mensagem é enviada pelo TE para o MT para requerer a transferência de uma mensagem SMS ou CBS específica armazenada no MT.

9. Delete Message - Esta mensagem é enviada pelo TE para o MT para requerer a eliminação de uma mensagem SMS ou CBS específica armazenada no MT.

10. Unable to Process - Esta mensagem é enviada pelo TE para o MT para indicar que a mensagem MT's não pode ser processada.

11. End SMS Mode - Esta mensagem é enviada pelo TE para o MT para terminar o modo SMS/CBS da interface DTE/DCE.

12. Acknowledge Message - Esta mensagem é enviada pelo TE para o MT para reconhecer a recepção de uma INC MESSAGE ou MESSAGE ARRIVED que contem um elemento identificador de uma SMS.

2.1.1.4.2. Respostas/indicações emitidas pelo MT:

1. Message List - Esta resposta é enviada do MT para o TE na recepção de LIST REQUEST proveniente do TE.

2. Message - Esta resposta é enviada do MT para o TE quando uma short message foi requerida

3. Get Message Failure - Esta resposta é enviada do MT para o TE quando o pedido de uma short message não pode ser cumprido.

4. Inc Message - Esta resposta é enviada do MT para o TE depois do MT ter pedido a transferência de mensagens de certas categorias imediatamente na recepção.

5. Message Arrived - Esta resposta é enviada do MT para o TE depois do MT ter pedido para providenciar uma indicação da recepção de certas categorias de mensagens 'incoming'

6. Insert SMS Complete - Esta resposta é enviada do MT para o TE para indicar que o pedido de TE´s para inserir uma mensagem foi completado.

7. Insert SMS Failure - Esta resposta é enviada do MT para o TE para indicar que a tentativa de inserir uma mensagem SMS falhou.

8. Delete Message Complete - Esta resposta é enviada do MT para o TE para indicar que o pedido de eliminação de uma mensagem da memória do MT está completo.

9. Delete Message Failure - Esta resposta é enviada do MT para o TE para indicar que o pedido de eliminação de uma mensagem falhou.

10. Unable to Process - Esta resposta é enviada do MT para o TE para indicar que o pedido de TE's não pode ser processado

11. End SMS Mode - Esta resposta é enviada do MT para o TE quando o MT sai independentemente do modo SMS/CBS.

12. Request Confirmed - Esta resposta é enviada do MT para o TE para indicar que o MT recebeu o pedido do TE e executará a função requerida.

2.1.1.5. Formato geral de mensagem e codificação dos elementos de informação

Este ponto descreve o conteúdo de mensagens para o modo SMS/CBS da interface DTE/DCE. Nas figuras seguintes, o bit designado por "bit 1" é transmitido primeiro, seguido dos bits 2, 3, 4, etc. Da mesma forma o octeto mostrado no topo de cada figura é enviado primeiro.

2.1.1.5.1. Tipo de Mensagem

O propósito do tipo de mensagem é identificar a função da mensagem a ser enviada. O bit 8 está reservado para um uso futuro como um bit de extensão.

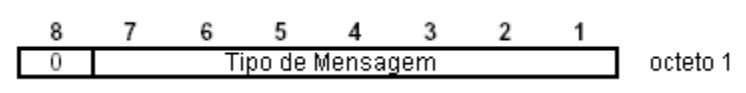


Fig.8 – Tipo de Mensagem

8	7	6	5	4	3	2	1	
0	0	0	-	-	-	-	-	Comandos/Respostas emitidos por TE
0	0	0	0	0	0	0	0	LIST REQUEST
0	0	0	0	0	0	0	1	GET MESSAGE
0	0	0	0	0	0	1	0	GET FIRST MESSAGE
0	0	0	0	0	0	1	1	GET NEXT MESSAGE
0	0	0	0	0	1	0	0	TRANSFER INC SMS
0	0	0	0	0	1	0	1	INDICATE INC SMS
0	0	0	0	0	1	1	0	TRANSFER INC CBS
0	0	0	0	0	1	1	1	INSERT SMS
0	0	0	0	1	0	0	0	DELETE MESSAGE
0	0	0	0	1	0	0	1	UNABLE TO PROCESS
0	0	0	1	1	1	1	0	END SMS MODE
0	0	0	1	1	1	1	1	ACKNOWLEDGE MESSAGE
0	0	1	-	-	-	-	-	Respostas/Indicações emitidas por MT
0	0	1	0	0	0	0	0	MESSAGE LIST
0	0	1	0	0	0	0	1	MESSAGE
0	0	1	0	0	0	1	0	GET MESSAGE FAILURE
0	0	1	0	0	0	1	1	INC MESSAGE
0	0	1	0	0	1	0	0	MESSAGE ARRIVED
0	0	1	0	0	1	0	1	INSERT SMS COMPLETE
0	0	1	0	0	1	1	0	INSERT SMS FAILURE
0	0	1	0	0	1	1	1	DELETE MESSAGE COMPLETE
0	0	1	0	1	0	0	0	DELETE MESSAGE FAILURE
0	0	1	0	1	0	0	1	UNABLE TO PROCESS
0	0	1	0	1	0	1	0	REQUEST CONFIRMED
0	0	1	1	1	1	1	1	END SMS MODE

Todos os outros valores estão reservados. Se um tipo de mensagem reservado é recebido, a identidade receptora deverá retornar 'Unable to Process' com a causa 'Command not understood'.

Fig.9 – Tipos de mensagem

2.2. Modo Texto

2.2.1 Descrição

Neste modo usamos a linha de comandos AT que consiste em:
Na rede GSM/UMTS existem regras de sintaxe de comandos estendidos, cada comando estendido tem um comando de teste para testar a existência do mesmo e para dar informação acerca dos seus subparâmetros.
Os comandos do "tipo de parâmetro (parameter type)" também têm o "comando ler (read command)" para verificar os valores dos subparâmetros.
Os "comandos do tipo acção (action type commands)" não armazenam os valores de nenhuns dos seus possíveis subparâmetros e, por isso, não têm o comando ler.

Exemplo:

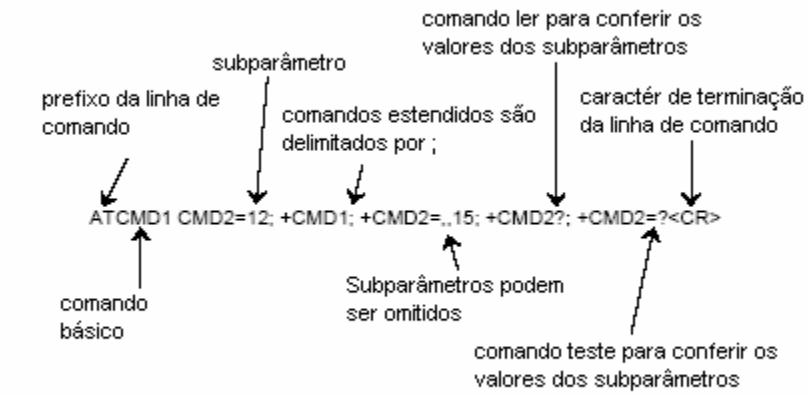


Fig.10-Estrutura básica de uma linha de comandos

Se as respostas são verbais (comando V1) e todos os comandos forem executados com sucesso, o código resultante <CR><LF> OK <CR><LF> são enviados do TA para o TE, caso os comandos não sejam executados com sucesso, o código resultante será <CR><LF> ERROR<CR><LF> e nenhum comando será processado.
Se as respostas forem numéricas (comando V0), e todos os comandos forem executados com sucesso, o código resultante será 0<CR> a ser enviado, caso os comandos não sejam executados com sucesso, o código resultante será 4<CR>.

Nota: A resposta ERROR ou 4 poderá ser substituída por +CME ERROR: <err> quando o comando não é processado devido a um erro relacionado com a operação do MT.

Legenda:
<CR> → carriage return (enter).
<LF> → line feed (mudança de linha).

2.2.1.2. Alguns comandos de configuração:

Comando	Objectivo
AT+CSMS	Escolher o serviço de mensagens
AT+CPMS	Escolher as memórias onde guardar e enviar mensagens
AT+CMGF	Escolher o modo de formato de mensagens (PDU, Texto)
AT+CESP	Escolher o modo Bloco
AT+CSCA	Escolher o serviço do centro de mensagens (“GSM, PCCP437, etc.)
AT+CSMP	Ajustar os parâmetros do modo texto
AT+CSDH	Mostrar parâmetros do modo texto
AT+CNMI	Indicar ao TE a chegada de novas mensagens
AT+CMGL	Listar as mensagens existentes na memória escolhida
AT+CMGR	Ler uma mensagem
AT+CNMA	Indica a chegada de nova mensagem ao ME/TA
AT+CMGS	Enviar mensagem
AT+CMSS	Enviar mensagem já existente no ME
AT+CMGW	Escrever mensagem na memória do ME
AT+CMGD	Apagar mensagem
AT+CMGC	Enviar uma mensagem de comando para a rede
AT+CMMS	Mais mensagens a enviar

2.2.1.3. Exemplo de utilização de um comando AT (para ler mensagem):

AT+CMGF=1<CR> → Escolhe o modo texto
OK
AT+CMGR=1<CR> → Ler a mensagem da posição 1 da memória escolhida
+CMGR: “REC READ”,”+351965554443”, ,”04/07/04,14:45:04+08”
Mensagem de teste
OK

2.2.1.4. Exemplo de utilização de um comando AT (para enviar mensagem):

AT+CMGF=1<CR> → Escolhe o modo texto
OK
AT+CMGS=”+351963456772” → Envia a mensagem para este número
>Mensagem de teste <CTRL+Z>
+CMGS: 6
OK

2.2.1.5. Resultado da programação implementada no VC++ para o envio de uma SMS no modo texto:
(Código em anexo)ficheiro 'modo_texto.cpp')

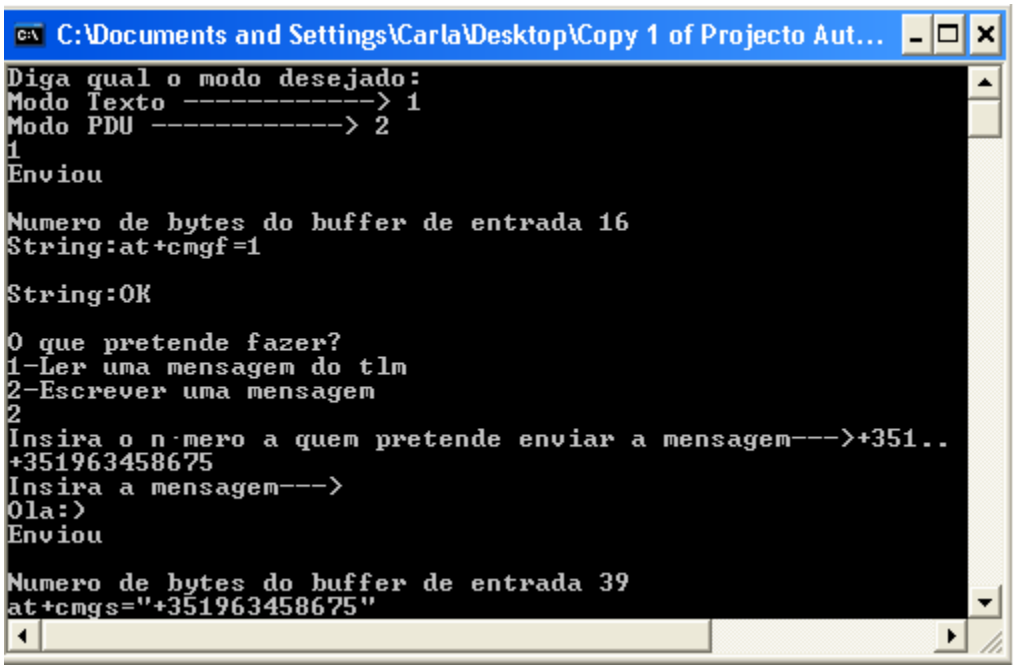


Fig.11 – Envio de uma SMS no modo texto

2.2.1.6. Resultado da programação implementada no VC++ para a recepção de uma SMS no modo texto:

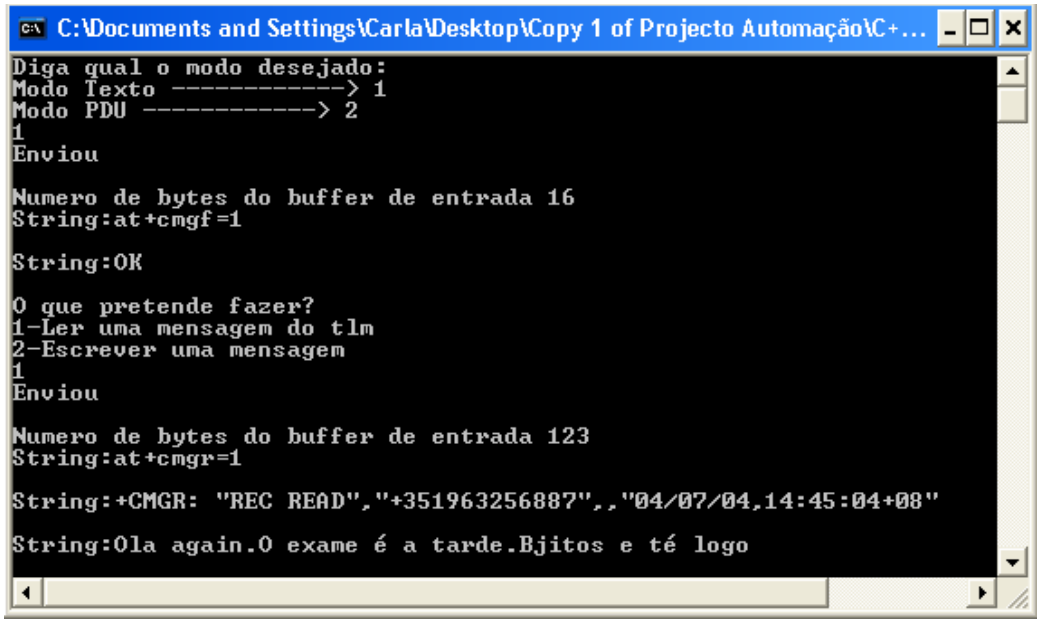


Fig.12 – Recepção de uma SMS no modo texto

2.3. Modo PDU

2.3.1. Recebendo uma mensagem no modo PDU
SMS-DELIVER (recebida no telemóvel)

Uma string PDU além de conter a mensagem, contém também uma série de informação sobre o remetente, como o Centro de serviço de SMS (SMSC), data e hora de envio, etc. Toda esta informação está em octetos hexa-decimal ou semi-octetos em decimal.

Exemplo de uma string recebida de um telemóvel (comando AT+CMGR=1\r):

07915391260100002 40C9153913652867800004070404154408 02C4E8329BFD4697D9EC77

Esta sequência de octetos consiste em 3 partes: um octeto inicial indicando o comprimento da informação do SMSC ("07"), a própria informação do SMSC ("91539126010000"), e a parte do SMS_DELIVER (remetente) ("240C…EC77").

Octeto(s)	Descrição
07	Comprimento da informação SMSC (neste caso 7 octetos)
91	Tipo de endereço do SMSC. (91 significa que está no formato internacional de nº de telemóvel)
539126010000	Nº do SMSC (em semi-octetos decimal). Neste caso o nº é 351962100000 (TMN)
24	1º octeto desta mensagem (SMS-DELIVER)
0C	Comprimento do nº do remetente (0C hex = 12 dec)
91	Tipo de endereço do nº do remetente
539136528678	Nº do remetente (semi-octetos em decimal)
00	TP-PID (Identificador do protocolo)
00	TP-DCS (Esquema de codificação da informação)
40704041544080	TP-SCTS (Data e hora de envio)
C4	TP-UDL (Comprimento da mensagem)
E8329BFD4697D9EC77	TP-UD (Mensagem "hellohello",em octetos de 8 bits representando mensagem de 7 bits) **(1)**

2.3.2. Enviando uma mensagem em modo PDU
SMS-SUBMIT (originada do telemóvel)

O exemplo seguinte mostra como enviar a mensagem “hellohello” no modo PDU.

AT+CMGF=0 //fixa o modo PDU
AT+CSMS=0 // Verifica se o modem suporta os comandos SMS
AT+CMGS=23 //Envia a SMS, 23 octetos (excluem-se os dois zeros iniciais)

>0011000C91539136528678 0000AA0AE8329BFD4697D9EC77 <CTRL+Z>

Octeto(s)	Descrição
00	Comprimento da informação SMSC. Neste caso ‘0’ octetos, o que significa que a informação SMSC armazenada no telemóvel é usada.
11	1º octeto desta mensagem SMS-SUBMIT
00	Referência-Mensagem-TP. O valor ‘00’ permite ao telemóvel fixar o número de referência de mensagem por si próprio
0C	Comprimento do nº de destino (0C hex = 12 dec)
91	Tipo de endereço do nº de destino (91 significa que está no formato internacional)
539136528678	Nº de destino (semi-octetos em decimal)
00	TP-PID (Identificador do protocolo)
00	TP-DCS (Esquema de codificação da informação)
AA	TP-Período-Validade
0A	TP-User-Data-Length (Comprimento da mensagem em caracteres/septetos)
E8329BFD4697D9EC77	TP-User-Data (Mensagem “hellohello”,em octetos de 8 bits representando mensagem de 7 bits) **(1)**

2.3.3. - (1) – Codificação de informação de 7-bit (septetos) em 8-bit (octetos)

A mensagem "hellohello" é formada por 10 caracteres, chamados septetos quando respresentados por 7 bits cada um. Estes septetos precisam de ser transformados em octetos para ser possível a transferência de SMS.

h	e	l	l	o	h	e	l	l	o
(a)104	101	108	108	111	104	101	108	108	111
(b)1101000	1100101	1101100	1101100	1101111	1101000	1100101	110100	1101100	1101111
1101000	1100101	1101100	1101100	1101111	1101000	1100101	110100	1101100	1101111

(a) – Código da tabela ascii dos caracteres em decimal
(b) – Código da tabela ascii dos caracteres em binário

O 1º septeto (h) é transformado num octeto, adicionando o bit mais significativo do 2º septeto. Este bit é inserido à esquerda 1 + 1101000 = 11101000 ("E8"). O bit mais significativo do 2º septeto é consumido, então o 2º septeto precisa de 2 bits do 3º caractere para fazer um octeto (8-bit). Este processo continua desta maneira progressivamente (bits a vermelho), resultando no seguinte:

11101000	00110010	10011011	11111101	01000110	10010111	1101001	1110100	1110111
E8	32	9B	FD	46	97	D9	EC	77

Então os 9 octetos de "hellohello" são: E8 32 9B FD 46 97 D9 EC 37

2.3.4. Resultado em VC++ da codificação de 7-bit para 8-bit
(ficheiro 'pduconv.c' em anexo)

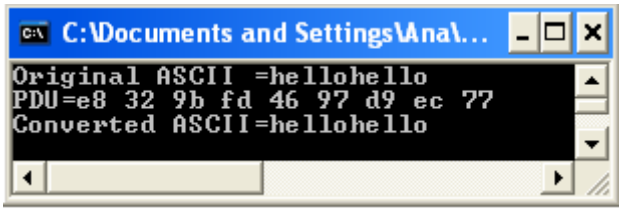


Fig.13 – Conversão de uma SMS para o modo PDU

2.3.5. Resultado em VC++ da programação das mensagens SMS-DELIVER e SMS_SUBMIT

A programação no VC++ das mensagens SMS-DELIVER e SMS_SUBMIT consiste na separação dos seus vários elementos, sendo o objectivo principal chegar à mensagem para ser posteriormente utilizada. (ficheiro 'decodPDU.c' em anexo)

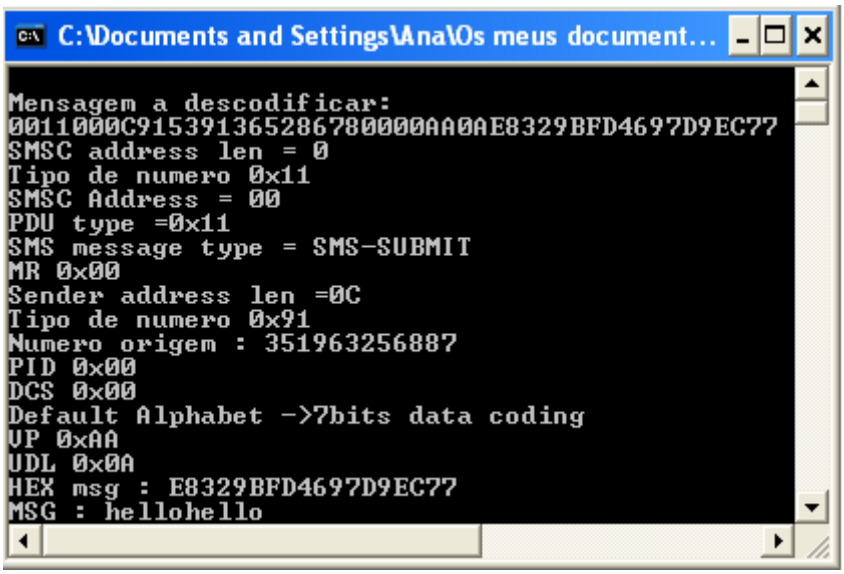


Fig.14 – Descodificação de uma mensagem do tipo SMS-SUBMIT

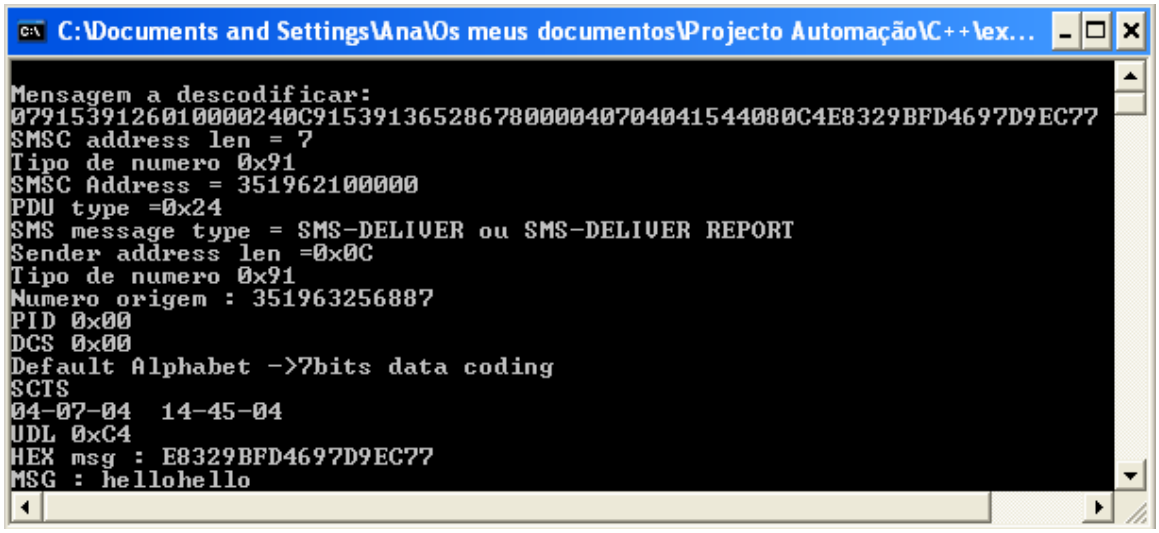


Fig.15 – Descodificação de uma mensagem do tipo SMS-DELIVER

3. Aplicação final

Para correr aplicação abrir o ficheiro **Proj_Sms.exe**. A porta série está configurada para **'COM1'**, se for necessário modificá-la, abrir o ficheiro (workspace) **Proj_SMS.dsw** (sendo necessário ter o VC++ 6.0 instalado), no **file view** abrir **tratar_tlm.cpp** e alterar a linha de código **COMPort aPort ("COM1");** para a porta desejada.
A interface da aplicação foi construída com a 'WIN 32 API do windows', tendo como resultado:
(código em anexo)

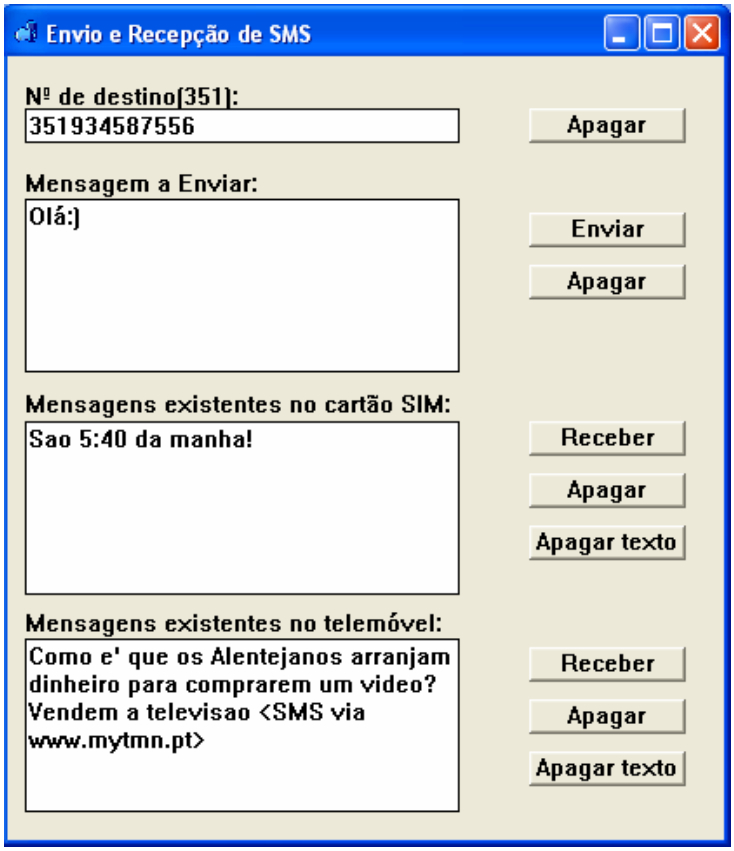


Fig.16 – Resultado da aplicação final

3.1. Protecções efectuadas

Nesta interface implementaram-se ainda protecções que resultam em erro, tais como:

- Se não for digitado nenhum número, nem mensagem.
- Se o nº estiver diferente de 35191,93,96 (formato português) e não tiver 12 algarismos.
- Se o telemóvel estiver inicialmente desligado.
- Se a mensagem possuir mais de 160 caracteres.
- Se ao receber não existe mensagem numa determinado índice do cartão SIM (capacidade para 10 SMS).

4. BIBLIOGRAFIA

- 23038-600.pdf - Norma GSM '3GPP TS 23.038' v.600'
- ts-127005v040200p.pdf - Norma GSM 'ETSI TS 127005 V.4.2.0'
- tutorial.pdf - The Forger's Win 32 API tutorial
- Charles Petzold – Programming Windows 5th edition.pdf
- SMS and the PDU format.pdf
- Sms-pdumode.pdf
- PDUCON-Convert between ASCII and PDU.html (função pduconv.c)
- Download C++ source code for win 32 API.html (função Comport.cpp para controlar a porta série)

Nota: Documentação no CD do projecto

5. ANEXOS

5.1. Ficheiro cod_descod.cpp

```
#include "stdio.h"
#include "conio.h"
#include "string.h"
//-----------------------------------------------------------------------------------------------
//Calculo do BCS
short int BCS(char str[],char size)
{
	int i=0;
	short int soma=0;

	for(i=0;i<=size;i++)
		soma=soma+str[i];
	soma=~soma+1;
	return soma; //devolve somente o BCS
}
//-----------------------------------------------------------------------------------------------
//BCS preparado
void smsBCS(char sms[],char BCS1[],int size)
{
	unsigned short int soma=0;
	char str3[256];
	int i=0,size2=0,j=0,k=0,l=0,f=0;

	soma=BCS(sms,size);		//Calculo do BCS da SMS
	BCS1[0]=soma / 256;		//parte mais significativa do BCS
	BCS1[1]=soma % 256;		//parte menos significativa do BCS

	printf("\n\nBCS preparado:\n");

	for(i=0;i<size;i++)
	{
		str3[f]=sms[i];
		printf(" %hX",str3[f]);
		f++;
	}
	for(l=1,j=size+3+l,k=0;j<size+3+3;j++,l++)  //junta o BCS a SMS original
	{
		printf(" %hX",BCS1[k]);
		k++;
	}
}
//-----------------------------------------------------------------------------------------------
//Insere os Octetos Stuffing na string da SMS

int stuff(char sms[],int size,char sms2[])
{
	int i=0,j=0;

	for(i=0;i<=size;i++)
	{
```

```
                sms2[j]=sms[i];

                if(sms[i]==0x10)// insere octeto de stuffing
                {
                        sms2[j+1]=0x00;
                        j++;
                }
                j++;
        }

        return j-1; //tamanho da sms + stuffs
}
//-----------------------------------------------------------------------------------------------
//Insere os Octetos Stuffing na string do BCS
int stuff2(char BCS1[],char BCS2[])
{
        int i=0,j=0;

        for(i=0;i<2;i++)
        {
                BCS2[j]=BCS1[i];

                if(BCS1[i]==0x10)// inserir octeto de stuffing
                {
                        BCS2[j+1]=0x00;
                        j++;

                }
                j++;
        }
        return j; //tamanho da str do BCS + stuffings
}
//-----------------------------------------------------------------------------------------------
//Adiciona marcas de Inicio e Fim
int marcas(char sms2[],char BCS2[],char sms3[],int tam_st,int tam_st2)
{
        sms3[0]=0x10;
        sms3[1]=0x02;
        int tam_sms=2;
        int i=0,j=0,k=0,u=2;

        // para inserir a marca de início
        for(j=2;j<tam_st+2;j++)
        {
                sms3[j]=sms2[i];
                i++;
                tam_sms++;
        }
        sms3[tam_sms]=0x10;    //insere marca de fim
        sms3[tam_sms+1]=0x03;  //insere marca de fim

        sms3[tam_sms+2]=BCS2[0];
        sms3[tam_sms+3]=BCS2[1];
        sms3[tam_sms+4]=BCS2[2];
        sms3[tam_sms+5]=BCS2[3];
```

```
	printf("\n\nMensagem como transmitida:\n");

	for(i=0;i<tam_st+4+tam_st2;i++)
		printf(" %hX",sms3[i]);

	return i; //tamanho da sms3
}
//-------------------------------------------------------------------------------------------
//resumo das funçoes anteriores na funçao envia
void envia(char sms[],int size)
{
	char BCS1[256];//str q resulta da funçao smsBCS
	char sms2[256];//str q resulta da função stuff (sms +stuffs)
	char BCS2[256];//str q resulta da f stuff2 (BCS+stuffs)
	char sms3[256]; //str q resulta da funçao marcas (sms+BCS+marcas)

	int i;
	int tam_st=0,tam_st2=0,tam_sms3=0;

	printf("Exemplo de Sms a ser enviada do TE para o MT:\n");
	for(i=0;i<size;i++)
		printf(" %hX",sms[i]);

	smsBCS(sms,BCS1,size); //sms + BCS (sms e BCS sao strings independentes)

	tam_st=stuff(sms,size,sms2); //insere stuffings na sms (sms+stuffs ->sms2)

	tam_st2=stuff2(BCS1,BCS2);		//insere stuffings no BCS (BCS+stuffs ->BCS2)

	tam_sms3=marcas(sms2,BCS2,sms3,tam_st,tam_st2); //junta sms+BCS+marcas ->(sms3)
}
//-------------------------------------------------------------------------------------------
//remove sttufings
int remove(int size,char sms3[],char sms4[])
{
	int i=0,j=0;
	int contstuff=0;

		for(i=0;i<size-1;i++)
		{
			if(sms3[i]==0x10 && sms3[i+1]==0x00 )// se octeto stuffing
			{
			sms4[j]=sms3[i];
			sms4[j+1]=sms3[i+2];
				if(sms3[i+2]==0x10&& sms3[i+3]==0x00){
					i=i+1;
					contstuff++;
					}
			i=i+2;
			j=j+2;

			contstuff++;
				}
			else
			{
			sms4[j]=sms3[i];
```

```
                        sms4[size-1-contstuff]=sms3[size-1];
                        j++;
                        }
                }
        printf("\n\nSaida no receptor (fase1):\n");

                for(i=0;i<size-contstuff;i++)
                printf(" %hX",sms4[i]);

        return contstuff;
}
//-------------------------------------------------------------------------------------------
//detecta marcas de inicio e fim
int detecta(int size,int size2,char sms4[],char sms5[])//entra sms4 e sai ((sms5)) sem
{
        //as marcas de inicio e fim
        int i=0,j=0;

        for(i=0;i<=size-size2;i++)
        {
                if(sms4[i]==0x10 && sms4[i+1]==0x02)// se marca início
                {
                                sms5[j]=sms4[i+2];
                                i=i+2;
                                j++;
                }

                else
                if(sms4[i]==0x10 && sms4[i+1]==0x03)
                {
                        sms5[j]=sms4[i+2];
                        sms5[j+1]=sms4[i+3];
                        i=i+2;
                   j=j+1;
                        }

                else
                        {
                                sms5[j]=sms4[i];
                                j++;
                        }
                }

        printf("\n\nSaida no receptor (fase1-b):\n");

        for(i=0;i<size-size2-4;i++)
                printf(" %hX",sms5[i]);


        return i;//tamanho da sms sem as marcas ((size3))

}


//-------------------------------------------------------------------------------------------
//Verifica BCS (0X)
```

```
void verifBCS(char sms5[],int size3)
{
	int i=0;
	int soma=0,soma1=0,soma2=0;

	for(i=0;i<size3-2;i++)
		soma=soma+sms5[i];

	soma1=256*(unsigned char)sms5[size3-2]+(unsigned char)sms5[size3-1];//operaçao inversa da f
	//(smsBCS) //na parte em que se separa o BCS em dois octetos

	soma2=(soma+soma1)%65536;

	printf("\n\nVerificao do BCS:");
	if (soma2==0)
	{
		printf("\nBCS correcto!!!  ( %hXX ) \n",soma2);
		printf("\nSaida no receptor (fase2)");
		for(i=0;i<size3-2;i++)
			printf(" %hX",sms5[i]);
	}
	else
		printf("\nBCS incorrecto!!!( %hX ) \n",soma2);

}

//-------------------------------------------------------------------------------------------
//funçao q recebe como parametro a sms do MT,atraves da funçao remove os stuffing sao retirados e
//as marcas de inicio e fim,finalmente verifica o BCS
void recebe(char sms2[],int size)
{
	char sms4[256];
	char sms5[256];
	int size2=0,size3=0,i=0;

	printf("\n\n<<<---------------------- DTE/DCE ---------------------->>>\n");

	printf("\nExemplo de Sms recebida, enviada a partir do MT:\n");
	for(i=0;i<size;i++)
		printf(" %hX",sms2[i]);

	size2=remove(size,sms2,sms4);//remove octetos stuffing
	//size2->n de octetos stuffing retirados

	size3=detecta(size,size2,sms4,sms5);
	//size3->tamanho da sms sem marcas

	verifBCS(sms5,size3);//verifica se o BCS esta correcto

}



//-------------------------------------------------------------------------------------------
void main(void)
```

```
{
        //exemplo de sms a ser enviada do TE(sms)
        char sms[256]={0x00,0x10,0x20,0x30,0x40,0x50};
        //ex de mensagem a ser recebida do MT (sms2)
        char sms2[256]={0x10,0x02,0x05,0x10,0x00,0x10,0x00,0x20,0x30,0x10,0x03,0xFF,0x8B};

        int size=6; //tamanho da sms emitida do TE ou MT

        envia(sms,size);

        recebe(sms2,13);

        getch();

}
```

5.2. Ficheiro m_texto.c

```
#include "comport.hpp"
#include<stdio.h>
#include<string.h>
//#include<stdlib.h>
#include<conio.h>
//#include<time.h>
#include<windows.h>
#include <malloc.h>
#include "pduconv.cpp"
#include "decodPDU.c"

COMPort aPort ("COM1");
//-------------------------------------------------------------------------------------------------------------------
// funcao que envia comandos AT

void enviaAT(char sms[])
{
    int i=0,est,cont=0;

// Para enviar caracteres
    while(sms[i]!=0)
    {
            aPort.write(sms[i]);

            i++;
            cont++;
    }
    printf("Enviou\n\n");


}

//-----------------------------------------------------------------------------------------
// funcao que envia mensagem

void enviaAT1(char sms[])
```

```
{
	int i=0,est,cont=0;

// Para enviar caracteres

	while(sms[i]!=0)
	{
		aPort.write(sms[i]);
		Sleep(30);
		i++;
		cont++;
	}
	printf("Enviou\n\n");
}
//---------------------------------------------------------------------------------------------
// funcao que envia comandos mensagens

void envia_msg(char numero[],char texto[])
{
	int i=0,est,j=0;

	char comando[512];

// comando todo a ser enviado --->at-numero e msg com ctrl Z de seguida
	sprintf(comando,"at+cmgw=\"%s\"\r%s\r%c",numero,texto,26);
	enviaAT1(comando);

//	printf("\nMessage -------->\n%s\nEnviou\n\n",comando);

}
//---------------------------------------------------------------------------------------------
// Para enviar mensagem escrita no computador
void envia_sms(char envia_sms1[])
{
	int i=0;
// Para enviar caracteres
	printf("Insira mensagem\n");
	do
	{
	aPort.write(envia_sms1[i]);
	i++;
	}while(envia_sms1[i]!=26);

// ctrl-Z ->26
//	aPort.EsperaPorto1();
}
//---------------------------------------------------------------------------------------------
// funcao para receber as respostas do MT

void recebeMT(char answer2[])
{
	int i=0,aux=0;

	for(i=0;i<400;i++){

		answer2[i]=	aPort.read();
```

```
                if(answer2[i]=='\n' ){
                        answer2[i+1]=0;

                        break;
                }
        }
}
//-------------------------------------------------------------------------------------------
// funcao para receber as respostas qnd se quer enviar sms
void recebeMT2(char answer3[])
{
      int i=0,aux=0;
      char envia_sms1[256];

      for(i=0;i<256;i++){

                answer3[i]=         aPort.read();
                printf("%c",answer3[i]);
                if(answer3[i]=='\n' ){
                        answer3[i+1]=0;

                        break;
                }
       }
}
//-----------------------------------------------------------------------
int decodPDU(char *sms);

//-------------------------------------------------------------------------------------------
void main (void)
{

      aPort.setBitRate (COMPort::br9600);
      aPort.setParity (COMPort::None);
      aPort.setDataBits (COMPort::db8);
      aPort.setStopBits (COMPort::sb1);
      aPort.setBlockingMode(10,10,10,100,1);

      int i=0,modo=0,est,n=0,j=0,modo2=0,num_tel=0;
      char answer1[256];
      char answer2[256];
      char answer3[256];
      char envia_sms1[256];
      char numero_tlf[12];
      char texto[256];
      int tam_sms3=0;
      int k=0;
      int len_centro_msg,len_origem_msg,tam_msg;


//exemplo de sms a ser enviada do TE
      unsigned char sms[256];


      printf("Diga qual o modo desejado:\n");
```

```
        printf("Modo Texto ------------> 1\n");
        printf("Modo PDU ------------> 2\n");
        scanf("%d",&modo);


    // para escolher se quer trabalhar em text mode ou pdu
        switch(modo)
        {
        case 1:   //modo texto
                        enviaAT("at+cmgf=1\r");
                        Sleep(1000);
                        est=aPort.VerificaPorto();
                        if(est!=0)
                        {
                                do{
                                recebeMT(answer2); //eco
                                printf("String:%s\n",answer2);
                                }while(strncmp(answer2,"OK",2)!=0 &&
    strncmp(answer2,"ERROR",5)!=05);

                        }
                        else
                                printf("Buffer vazio\n");

                        printf("O que pretende fazer?\n");
                        printf("1-Ler uma mensagem do tlm\n");
                        printf("2-Escrever uma mensagem\n");
                        scanf("%d",&modo2);
                        break;

        case 2:   //modo pdu
                        enviaAT("at+cmgf=0\r");
                        Sleep(1000);
                        est=aPort.VerificaPorto();


                        do{
                        recebeMT(answer2); //eco

                        printf("String:%s\n",answer2);
                        }while(strncmp(answer2,"OK",2)!=0);



                        enviaAT("at+cmgr=1\r");
                        printf("OK\n");

                        Sleep(1000);


                        est=aPort.VerificaPorto();

                        for(i=0;i<2;i++)
                        {
```

```
                        if(est!=0){
                                recebeMT(answer2); //eco
                                printf("String:%s\n",answer2);
                        }

                        else{
                                printf("Buffer vazio\n");
                                break;
                        }
                } //ignora o que recebeu



                recebeMT(answer2); //Mensagem
                decodPDU(answer2);

                printf("\n");

                do{
                        recebeMT(answer2); //eco
                        printf("String:%s\n",answer2);
                        k++;
                        }while(k<3);
                break;

default:printf("Numero incorrecto\n");
                break;
        }

//escolher dentro do modo texto o que pretende fazer

switch(modo2)
{
case 1:   //ler sms do tlm
                enviaAT("at+cmgr=1\r");
                Sleep(1000);
                est=aPort.VerificaPorto();
                if(est!=0)
                {
                        do{
                        recebeMT(answer2); //eco
                        printf("String:%s\n",answer2);
                        }while(strncmp(answer2,">",1)!=0);
                }
                else
                        printf("Buffer vazio\n");
                break;

case 2:   //enviar sms

                printf("Insira o número a quem pretende enviar a mensagem--->+351..\n");

                //for(i=0;i<14;i++)
                //        scanf("%c",&numero_tlf[i]);

                fflush(stdin);
```

```
                        gets(numero_tlf);
                        fflush(stdin);//limpa buffer
                        printf("Insira a mensagem--->\n");
                        gets(texto);
                        printf("blabla:  %s\n%s\n",numero_tlf,texto);

                        envia_msg(numero_tlf,texto);

                        Sleep(100);
                        est=aPort.VerificaPorto();

                        if(est!=0)
                        {

                                do
                                {
                                        recebeMT2(answer3); //eco
                                        printf("String:%s\n",answer3);

                                }while(strncmp(answer2,"OK",2)!=0);

                        }//fecha if
                        else
                                printf("Buffer vazio\n");



                        break;

        default:printf("Número incorrecto\n");
                        break;
        }


                printf("\nEsperou\n");

                aPort.closeport();
                getch();

} //fim
```

5.3. Ficheiro pduconv.c

```
/***************************************************************************
 *
 * Functions for converting between an ISO-8859-1 ASCII string and a
 * PDU-coded string as described in ETSI GSM 03.38 and ETSI GSM 03.40.
 *
 * This code is released to the public domain in 2003 by Mats Engstrom,
 * Nerdlabs Consulting. ( matseng at nerdlabs dot org )
 *
 ***************************************************************************/

#include <string.h>
#include <stdlib.h>
```

```
#include <malloc.h>

#include "pduconv.h"

#define VERSION 0.1

/* Define Non-Printable Characters as a question mark */
#define NPC7    63
#define NPC8    '?'

/*****************************************************************************
   This lookup table converts from ISO-8859-1 8-bit ASCII to the
   7 bit "default alphabet" as defined in ETSI GSM 03.38

   ISO-characters that don't have any correspondning character in the
   7-bit alphabet is replaced with the NPC7-character.  If there's
   a close match between the ISO-char and a 7-bit character (for example
   the letter i with a circumflex and the plain i-character) a substitution
   is done. These "close-matches" are marked in the lookup table by
   having its value negated.

   There are some character (for example the curly brace "}") that must
   be converted into a 2 byte 7-bit sequence.  These characters are
   marked in the table by having 256 added to its value.
*****************************************************************************/

int lookup_ascii8to7[]={
   NPC7,      /*    0     null [NUL]                    */
   NPC7,      /*    1     start of heading [SOH]          */
   NPC7,      /*    2     start of text [STX]             */
   NPC7,      /*    3     end of text [ETX]               */
   NPC7,      /*    4     end of transmission [EOT]       */
   NPC7,      /*    5     enquiry [ENQ]                   */
   NPC7,      /*    6     acknowledge [ACK]               */
   NPC7,      /*    7     bell [BEL]                      */
   NPC7,      /*    8     backspace [BS]                  */
   NPC7,      /*    9     horizontal tab [HT]             */
   10,       /*   10     line feed [LF]                  */
   NPC7,      /*   11     vertical tab [VT]               */
   10+256,    /*   12     form feed [FF]                  */
   13,       /*   13     carriage return [CR]            */
   NPC7,      /*   14     shift out [SO]                  */
   NPC7,      /*   15     shift in [SI]                   */
   NPC7,      /*   16     data link escape [DLE]          */
   NPC7,      /*   17     device control 1 [DC1]          */
   NPC7,      /*   18     device control 2 [DC2]          */
   NPC7,      /*   19     device control 3 [DC3]          */
   NPC7,      /*   20     device control 4 [DC4]          */
   NPC7,      /*   21     negative acknowledge [NAK]      */
   NPC7,      /*   22     synchronous idle [SYN]          */
   NPC7,      /*   23     end of trans. block [ETB]       */
   NPC7,      /*   24     cancel [CAN]                    */
   NPC7,      /*   25     end of medium [EM]              */
   NPC7,      /*   26     substitute [SUB]                */
   NPC7,      /*   27     escape [ESC]                    */
   NPC7,      /*   28     file separator [FS]             */
```

```
NPC7,     /*  29    group separator [GS]          */
NPC7,     /*  30    record separator [RS]         */
NPC7,     /*  31    unit separator [US]           */
32,       /*  32    space                         */
33,       /*  33    ! exclamation mark            */
34,       /*  34    " double quotation mark       */
35,       /*  35    # number sign                 */
2,        /*  36    $ dollar sign                 */
37,       /*  37    % percent sign                */
38,       /*  38    & ampersand                   */
39,       /*  39    ' apostrophe                  */
40,       /*  40    ( left parenthesis            */
41,       /*  41    ) right parenthesis           */
42,       /*  42    * asterisk                    */
43,       /*  43    + plus sign                   */
44,       /*  44    , comma                       */
45,       /*  45    - hyphen                      */
46,       /*  46    . period                      */
47,       /*  47    / slash,                      */
48,       /*  48    0 digit 0                     */
49,       /*  49    1 digit 1                     */
50,       /*  50    2 digit 2                     */
51,       /*  51    3 digit 3                     */
52,       /*  52    4 digit 4                     */
53,       /*  53    5 digit 5                     */
54,       /*  54    6 digit 6                     */
55,       /*  55    7 digit 7                     */
56,       /*  56    8 digit 8                     */
57,       /*  57    9 digit 9                     */
58,       /*  58    : colon                       */
59,       /*  59    ; semicolon                   */
60,       /*  60    < less-than sign              */
61,       /*  61    = equal sign                  */
62,       /*  62    > greater-than sign           */
63,       /*  63    ? question mark               */
0,        /*  64    @ commercial at sign          */
65,       /*  65    A uppercase A                 */
66,       /*  66    B uppercase B                 */
67,       /*  67    C uppercase C                 */
68,       /*  68    D uppercase D                 */
69,       /*  69    E uppercase E                 */
70,       /*  70    F uppercase F                 */
71,       /*  71    G uppercase G                 */
72,       /*  72    H uppercase H                 */
73,       /*  73    I uppercase I                 */
74,       /*  74    J uppercase J                 */
75,       /*  75    K uppercase K                 */
76,       /*  76    L uppercase L                 */
77,       /*  77    M uppercase M                 */
78,       /*  78    N uppercase N                 */
79,       /*  79    O uppercase O                 */
80,       /*  80    P uppercase P                 */
81,       /*  81    Q uppercase Q                 */
82,       /*  82    R uppercase R                 */
83,       /*  83    S uppercase S                 */
84,       /*  84    T uppercase T                 */
```

```
85,       /*   85   U uppercase U                    */
86,       /*   86   V uppercase V                    */
87,       /*   87   W uppercase W                     */
88,       /*   88   X uppercase X                    */
89,       /*   89   Y uppercase Y                    */
90,       /*   90   Z uppercase Z                    */
60+256,    /*   91   [ left square bracket              */
47+256,    /*   92   \ backslash                      */
62+256,    /*   93   ] right square bracket              */
20+256,    /*   94   ^ circumflex accent                 */
17,       /*   95   _ underscore                    */
-39,       /*   96   ` back apostrophe                 */
97,       /*   97   a lowercase a                    */
98,       /*   98   b lowercase b                    */
99,       /*   99   c lowercase c                    */
100,      /*   100   d lowercase d                    */
101,      /*   101   e lowercase e                    */
102,      /*   102   f lowercase f                    */
103,      /*   103   g lowercase g                    */
104,      /*   104   h lowercase h                    */
105,      /*   105   i lowercase i                    */
106,      /*   106   j lowercase j                    */
107,      /*   107   k lowercase k                    */
108,      /*   108   l lowercase l                    */
109,      /*   109   m lowercase m                       */
110,      /*   110   n lowercase n                    */
111,      /*   111   o lowercase o                    */
112,      /*   112   p lowercase p                    */
113,      /*   113   q lowercase q                    */
114,      /*   114   r lowercase r                    */
115,      /*   115   s lowercase s                    */
116,      /*   116   t lowercase t                    */
117,      /*   117   u lowercase u                    */
118,      /*   118   v lowercase v                    */
119,      /*   119   w lowercase w                     */
120,      /*   120   x lowercase x                    */
121,      /*   121   y lowercase y                    */
122,      /*   122   z lowercase z                    */
40+256,    /*   123   { left brace                    */
64+256,    /*   124   | vertical bar                  */
41+256,    /*   125   } right brace                    */
61+256,    /*   126   ~ tilde accent                   */
NPC7,     /*   127    delete [DEL]                    */
NPC7,     /*   128                                 */
NPC7,     /*   129                                 */
-39,      /*   130    low left rising single quote          */
-102,     /*   131    lowercase italic f                 */
-34,      /*   132    low left rising double quote          */
NPC7,     /*   133    low horizontal ellipsis              */
NPC7,     /*   134    dagger mark                    */
NPC7,     /*   135    double dagger mark                 */
NPC7,     /*   136    letter modifying circumflex            */
NPC7,     /*   137    per thousand (mille) sign             */
-83,      /*   138    uppercase S caron or hacek            */
-39,      /*   139    left single angle quote mark          */
-214,     /*   140    uppercase OE ligature                */
```

```
NPC7,     /*  141                                 */
NPC7,     /*  142                                 */
NPC7,     /*  143                                 */
NPC7,     /*  144                                 */
-39,      /*  145    left single quotation mark   */
-39,      /*  146    right single quote mark      */
-34,      /*  147    left double quotation mark   */
-34,      /*  148    right double quote mark      */
-42,      /*  149    round filled bullet          */
-45,      /*  150    en dash                      */
-45,      /*  151    em dash                      */
-39,      /*  152    small spacing tilde accent   */
NPC7,     /*  153     trademark sign              */
-115,     /*  154     lowercase s caron or hacek  */
-39,      /*  155    right single angle quote mark*/
-111,     /*  156     lowercase oe ligature       */
NPC7,     /*  157                                 */
NPC7,     /*  158                                 */
-89,      /*  159    uppercase Y dieresis or umlaut*/
-32,      /*  160    non-breaking space           */
64,       /*  161   ¡ inverted exclamation mark   */
-99,      /*  162   ¢ cent sign                   */
1,        /*  163   £ pound sterling sign         */
36,       /*  164   ¤ general currency sign       */
3,        /*  165   ¥ yen sign                    */
-33,      /*  166   ¦ broken vertical bar         */
95,       /*  167   § section sign                */
-34,      /*  168   ¨ spacing dieresis or umlaut  */
NPC7,     /*  169   © copyright sign              */
NPC7,     /*  170   ª feminine ordinal indicator  */
-60,      /*  171   « left (double) angle quote   */
NPC7,     /*  172   ¬ logical not sign            */
-45,      /*  173   - soft hyphen                 */
NPC7,     /*  174   ® registered trademark sign   */
NPC7,     /*  175   ¯ spacing macron (long) accent*/
NPC7,     /*  176   ° degree sign                 */
NPC7,     /*  177   ± plus-or-minus sign          */
-50,      /*  178   ² superscript 2               */
-51,      /*  179   ³ superscript 3               */
-39,      /*  180   ´ spacing acute accent        */
-117,     /*  181   µ micro sign                  */
NPC7,     /*  182   ¶ paragraph sign, pilcrow sign*/
NPC7,     /*  183   · middle dot, centered dot    */
NPC7,     /*  184   ¸ spacing cedilla             */
-49,      /*  185   ¹ superscript 1               */
NPC7,     /*  186   º masculine ordinal indicator */
-62,      /*  187   » right (double) angle quote (guillemet) */
NPC7,     /*  188   ¼ fraction 1/4                */
NPC7,     /*  189   ½ fraction 1/2                */
NPC7,     /*  190   ¾ fraction 3/4                */
96,       /*  191   ¿ inverted question mark      */
-65,      /*  192   À uppercase A grave           */
-65,      /*  193   Á uppercase A acute           */
-65,      /*  194   Â uppercase A circumflex       */
-65,      /*  195   Ã uppercase A tilde            */
91,       /*  196   Ä uppercase A dieresis or umlaut */
```

```
14,     /*  197   Å uppercase A ring                */
28,     /*  198   Æ uppercase AE ligature           */
9,      /*  199   Ç uppercase C cedilla             */
-31,    /*  200   È uppercase E grave               */
31,     /*  201   É uppercase E acute               */
-31,    /*  202   Ê uppercase E circumflex          */
-31,    /*  203   Ë uppercase E dieresis or umlaut  */
-73,    /*  204   Ì uppercase I grave               */
-73,    /*  205   Í uppercase I acute               */
-73,    /*  206   Î uppercase I circumflex          */
-73,    /*  207   Ï uppercase I dieresis or umlaut  */
-68,    /*  208   Ð uppercase ETH                   */
93,     /*  209   Ñ uppercase N tilde               */
-79,    /*  210   Ò uppercase O grave               */
-79,    /*  211   Ó uppercase O acute               */
-79,    /*  212   Ô uppercase O circumflex          */
-79,    /*  213   Õ uppercase O tilde               */
92,     /*  214   Ö uppercase O dieresis or umlaut  */
-42,    /*  215   × multiplication sign             */
11,     /*  216   Ø uppercase O slash               */
-85,    /*  217   Ù uppercase U grave               */
-85,    /*  218   Ú uppercase U acute               */
-85,    /*  219   Û uppercase U circumflex          */
94,     /*  220   Ü uppercase U dieresis or umlaut  */
-89,    /*  221   Ý uppercase Y acute               */
NPC7,   /*  222   Þ uppercase THORN                 */
30,     /*  223   ß lowercase sharp s, sz ligature  */
127,    /*  224   à lowercase a grave               */
-97,    /*  225   á lowercase a acute               */
-97,    /*  226   â lowercase a circumflex          */
-97,    /*  227   ã lowercase a tilde               */
123,    /*  228   ä lowercase a dieresis or umlaut  */
15,     /*  229   å lowercase a ring                */
29,     /*  230   æ lowercase ae ligature           */
-9,     /*  231   ç lowercase c cedilla             */
4,      /*  232   è lowercase e grave               */
5,      /*  233   é lowercase e acute               */
-101,   /*  234   ê lowercase e circumflex          */
-101,   /*  235   ë lowercase e dieresis or umlaut  */
7,      /*  236   ì lowercase i grave               */
7,      /*  237   í lowercase i acute               */
-105,   /*  238   î lowercase i circumflex          */
-105,   /*  239   ï lowercase i dieresis or umlaut  */
NPC7,   /*  240   ð lowercase eth                   */
125,    /*  241   ñ lowercase n tilde               */
8,      /*  242   ò lowercase o grave               */
-111,   /*  243   ó lowercase o acute               */
-111,   /*  244   ô lowercase o circumflex          */
-111,   /*  245   õ lowercase o tilde               */
124,    /*  246   ö lowercase o dieresis or umlaut  */
-47,    /*  247   ÷ division sign                   */
12,     /*  248   ø lowercase o slash               */
6,      /*  249   ù lowercase u grave               */
-117,   /*  250   ú lowercase u acute               */
-117,   /*  251   û lowercase u circumflex          */
126,    /*  252   ü lowercase u dieresis or umlaut  */
```

```
   -121,      /*   253    ý lowercase y acute                    */
   NPC7,       /*   254    þ lowercase thorn                      */
   -121        /*   255    ÿ lowercase y dieresis or umlaut       */
};



/*****************************************************************************
  This lookup table converts from the 7 bit "default alphabet" as
   defined in ETSI GSM 03.38 to a standard ISO-8859-1 8-bit ASCII.

   Some characters in the 7-bit alphabet does not exist in the ISO
   character set, they are replaced by the NPC8-character.

   If the character is decimal 27 (ESC) the following character have
   a special meaning and must be handled separately.
*****************************************************************************/

int lookup_ascii7to8[]={
   64,        /*  0     @  COMMERCIAL AT                        */
   163,       /*  1     £  POUND SIGN                           */
   36,        /*  2     $  DOLLAR SIGN                          */
   165,       /*  3     ¥  YEN SIGN                             */
   232,       /*  4     è  LATIN SMALL LETTER E WITH GRAVE      */
   233,       /*  5     é  LATIN SMALL LETTER E WITH ACUTE      */
   249,       /*  6     ù  LATIN SMALL LETTER U WITH GRAVE      */
   236,       /*  7     ì  LATIN SMALL LETTER I WITH GRAVE      */
   242,       /*  8     ò  LATIN SMALL LETTER O WITH GRAVE      */
   199,       /*  9     Ç  LATIN CAPITAL LETTER C WITH CEDILLA  */
   10,        /*  10      LINE FEED                             */
   216,       /*  11    Ø  LATIN CAPITAL LETTER O WITH STROKE   */
   248,       /*  12    ø  LATIN SMALL LETTER O WITH STROKE     */
   13,        /*  13      CARRIAGE RETURN                       */
   197,       /*  14    Å  LATIN CAPITAL LETTER A WITH RING ABOVE */
   229,       /*  15    å  LATIN SMALL LETTER A WITH RING ABOVE  */
   NPC8,      /*  16      GREEK CAPITAL LETTER DELTA            */
   95,        /*  17    _  LOW LINE                             */
   NPC8,      /*  18      GREEK CAPITAL LETTER PHI              */
   NPC8,      /*  19      GREEK CAPITAL LETTER GAMMA            */
   NPC8,      /*  20      GREEK CAPITAL LETTER LAMBDA           */
   NPC8,      /*  21      GREEK CAPITAL LETTER OMEGA            */
   NPC8,      /*  22      GREEK CAPITAL LETTER PI               */
   NPC8,      /*  23      GREEK CAPITAL LETTER PSI              */
   NPC8,      /*  24      GREEK CAPITAL LETTER SIGMA            */
   NPC8,      /*  25      GREEK CAPITAL LETTER THETA            */
   NPC8,      /*  26      GREEK CAPITAL LETTER XI               */
   27,        /*  27      ESCAPE TO EXTENSION TABLE             */
   198,       /*  28    Æ  LATIN CAPITAL LETTER AE              */
   230,       /*  29    æ  LATIN SMALL LETTER AE                */
   223,       /*  30    ß  LATIN SMALL LETTER SHARP S (German)  */
   201,       /*  31    É  LATIN CAPITAL LETTER E WITH ACUTE    */
   32,        /*  32      SPACE                                 */
   33,        /*  33    !  EXCLAMATION MARK                     */
   34,        /*  34    "  QUOTATION MARK                       */
   35,        /*  35    #  NUMBER SIGN                          */
   164,       /*  36    ¤  CURRENCY SIGN                        */
```

```
37,     /*  37   %  PERCENT SIGN                    */
38,     /*  38   &  AMPERSAND                       */
39,     /*  39   '  APOSTROPHE                      */
40,     /*  40   (  LEFT PARENTHESIS                */
41,     /*  41   )  RIGHT PARENTHESIS               */
42,     /*  42   *  ASTERISK                        */
43,     /*  43   +  PLUS SIGN                       */
44,     /*  44   ,  COMMA                           */
45,     /*  45   -  HYPHEN-MINUS                    */
46,     /*  46   .  FULL STOP                       */
47,     /*  47   /  SOLIDUS (SLASH)                 */
48,     /*  48   0  DIGIT ZERO                      */
49,     /*  49   1  DIGIT ONE                       */
50,     /*  50   2  DIGIT TWO                       */
51,     /*  51   3  DIGIT THREE                     */
52,     /*  52   4  DIGIT FOUR                      */
53,     /*  53   5  DIGIT FIVE                      */
54,     /*  54   6  DIGIT SIX                       */
55,     /*  55   7  DIGIT SEVEN                     */
56,     /*  56   8  DIGIT EIGHT                     */
57,     /*  57   9  DIGIT NINE                      */
58,     /*  58   :  COLON                           */
59,     /*  59   ;  SEMICOLON                       */
60,     /*  60   <  LESS-THAN SIGN                  */
61,     /*  61   =  EQUALS SIGN                     */
62,     /*  62   >  GREATER-THAN SIGN               */
63,     /*  63   ?  QUESTION MARK                   */
161,    /*  64   ¡  INVERTED EXCLAMATION MARK       */
65,     /*  65   A  LATIN CAPITAL LETTER A          */
66,     /*  66   B  LATIN CAPITAL LETTER B          */
67,     /*  67   C  LATIN CAPITAL LETTER C          */
68,     /*  68   D  LATIN CAPITAL LETTER D          */
69,     /*  69   E  LATIN CAPITAL LETTER E          */
70,     /*  70   F  LATIN CAPITAL LETTER F          */
71,     /*  71   G  LATIN CAPITAL LETTER G          */
72,     /*  72   H  LATIN CAPITAL LETTER H          */
73,     /*  73   I  LATIN CAPITAL LETTER I          */
74,     /*  74   J  LATIN CAPITAL LETTER J          */
75,     /*  75   K  LATIN CAPITAL LETTER K          */
76,     /*  76   L  LATIN CAPITAL LETTER L          */
77,     /*  77   M  LATIN CAPITAL LETTER M          */
78,     /*  78   N  LATIN CAPITAL LETTER N          */
79,     /*  79   O  LATIN CAPITAL LETTER O          */
80,     /*  80   P  LATIN CAPITAL LETTER P          */
81,     /*  81   Q  LATIN CAPITAL LETTER Q          */
82,     /*  82   R  LATIN CAPITAL LETTER R          */
83,     /*  83   S  LATIN CAPITAL LETTER S          */
84,     /*  84   T  LATIN CAPITAL LETTER T          */
85,     /*  85   U  LATIN CAPITAL LETTER U          */
86,     /*  86   V  LATIN CAPITAL LETTER V          */
87,     /*  87   W  LATIN CAPITAL LETTER W          */
88,     /*  88   X  LATIN CAPITAL LETTER X          */
89,     /*  89   Y  LATIN CAPITAL LETTER Y          */
90,     /*  90   Z  LATIN CAPITAL LETTER Z          */
196,    /*  91   Ä  LATIN CAPITAL LETTER A WITH DIAERESIS   */
214,    /*  92   Ö  LATIN CAPITAL LETTER O WITH DIAERESIS   */
```

```
    209,      /*  93    Ñ  LATIN CAPITAL LETTER N WITH TILDE      */
    220,      /*  94    Ü  LATIN CAPITAL LETTER U WITH DIAERESIS   */
    167,      /*  95    §  SECTION SIGN                  */
    191,      /*  96    ¿  INVERTED QUESTION MARK              */
    97,      /*  97    a  LATIN SMALL LETTER A              */
    98,      /*  98    b  LATIN SMALL LETTER B              */
    99,      /*  99    c  LATIN SMALL LETTER C              */
    100,      /*  100   d  LATIN SMALL LETTER D               */
    101,      /*  101   e  LATIN SMALL LETTER E               */
    102,      /*  102   f  LATIN SMALL LETTER F               */
    103,      /*  103   g  LATIN SMALL LETTER G               */
    104,      /*  104   h  LATIN SMALL LETTER H               */
    105,      /*  105   i  LATIN SMALL LETTER I              */
    106,      /*  106   j  LATIN SMALL LETTER J              */
    107,      /*  107   k  LATIN SMALL LETTER K               */
    108,      /*  108   l  LATIN SMALL LETTER L               */
    109,      /*  109   m  LATIN SMALL LETTER M               */
    110,      /*  110   n  LATIN SMALL LETTER N               */
    111,      /*  111   o  LATIN SMALL LETTER O               */
    112,      /*  112   p  LATIN SMALL LETTER P               */
    113,      /*  113   q  LATIN SMALL LETTER Q               */
    114,      /*  114   r  LATIN SMALL LETTER R               */
    115,      /*  115   s  LATIN SMALL LETTER S               */
    116,      /*  116   t  LATIN SMALL LETTER T               */
    117,      /*  117   u  LATIN SMALL LETTER U               */
    118,      /*  118   v  LATIN SMALL LETTER V               */
    119,      /*  119   w  LATIN SMALL LETTER W               */
    120,      /*  120   x  LATIN SMALL LETTER X               */
    121,      /*  121   y  LATIN SMALL LETTER Y               */
    122,      /*  122   z  LATIN SMALL LETTER Z              */
    228,      /*  123   ä  LATIN SMALL LETTER A WITH DIAERESIS    */
    246,      /*  124   ö  LATIN SMALL LETTER O WITH DIAERESIS    */
    241,      /*  125   ñ  LATIN SMALL LETTER N WITH TILDE       */
    252,      /*  126   ü  LATIN SMALL LETTER U WITH DIAERESIS    */
    224       /*  127   à  LATIN SMALL LETTER A WITH GRAVE       */

  /*  The double bytes below must be handled separately after the
     table lookup.

     12          27 10     FORM FEED
     94          27 20   ^  CIRCUMFLEX ACCENT
     123          27 40   {  LEFT CURLY BRACKET
     125          27 41   }  RIGHT CURLY BRACKET
     92          27 47   \  REVERSE SOLIDUS (BACKSLASH)
     91          27 60   [  LEFT SQUARE BRACKET
     126          27 61   ~  TILDE
     93          27 62   ]  RIGHT SQUARE BRACKET
     124          27 64   |  VERTICAL BAR                    */

  };
  /*
   * Use a lookup table to convert from the 7-bit default alphabet
   * used by SMS to an ISO-8859-1 ASCII string.
   *
   *  *a7bit   An array of the 7-bit 'string' to convert
   *
```

```
 *  length   The length of the a7bit-array
 *
 *  **ascii  A pointer to the string that the result is stored in,
 *           the space is malloced by this function
 *
 *  Returns the length of the ascii-string
 *
 *  Note:  The ascii-string must be free()'ed by te caller
 */

static int convert_7bit_to_ascii(char *a7bit, int length, char **ascii) {
   int     r;
   int     w;

   /* Allocate sufficient memory for the result string */
   *ascii=malloc(length*2+1);

   w=0;
   for (r=0; r<length; r++) {
      if ((lookup_ascii7to8[(unsigned char)a7bit[r]])!=27) {
         (*ascii)[w++]=lookup_ascii7to8[(unsigned char)a7bit[r]];
      } else {
         /* If we're escaped then the next byte have a special meaning. */
         r++;
         switch (a7bit[r]) {
            case    10:
               (*ascii)[w++]=12;
               break;
            case    20:
               (*ascii)[w++]='^';
               break;
            case    40:
               (*ascii)[w++]='{';
               break;
            case    41:
               (*ascii)[w++]='}';
               break;
            case    47:
               (*ascii)[w++]='\\';
               break;
            case    60:
               (*ascii)[w++]='[';
               break;
            case    61:
               (*ascii)[w++]='~';
               break;
            case    62:
               (*ascii)[w++]=']';
               break;
           case    64:
               (*ascii)[w++]='|';
               break;
            default:
               (*ascii)[w++]=NPC8;
               break;
         }
```

```
        }
    }

    /* Terminate the result string and realloc to the correct size */
    (*ascii)[w]=0;
    *ascii=realloc(*ascii,w+1);

    return w;
}



/*
 * Use a lookup table to convert from an ISO-8859-1 string to
 * the 7-bit default alphabet used by SMS.
 *
 *  *scii   The string to convert
 *
 *  **a7bit  A pointer to the array that the result is stored in,
 *           the space is malloced by this function
 *
 *  Returns the length of the a7bit-array
 *
 *  Note:  The a7bit-array must be free()'ed by te caller
 */

static int convert_ascii_to_7bit(char *ascii, char **a7bit) {
    int r;
    int w;

    r=0;
    w=0;

    /* Allocate sufficient memory for the result string */
    *a7bit=malloc(strlen(ascii)*2+1);

    while (ascii[r]!=0) {
        if ((lookup_ascii8to7[(unsigned char)ascii[r]])<256) {
            (*a7bit)[w++]=abs(lookup_ascii8to7[(unsigned char)ascii[r++]]);
        } else {
            (*a7bit)[w++]=27;
            (*a7bit)[w++]=lookup_ascii8to7[(unsigned char)ascii[r++]]-256;
        }
    }

    /* Realloc the result array to the correct size */
    *a7bit=realloc(*a7bit,w);

    return w;
}

/*
 *  Convert a PDU-coded string to ISO-8859-1 ASCII
 *
 *  *pdu      The pdu-array to convert to cleartext
 *
```

```
 *  pdulength The length of the pdu-array
 *
 *  **ascii   A pointer to the buffer that the cleartext will be written to,
 *            the space for the buffer will be allocated by this function
 *
 *  Returns the length of the ascii-string
 *
 *  Note:  Don't forget to free() the ascii-string when you're finished
 *         with it.
 *
 */

int pdu_to_ascii(unsigned char *pdu, int pdulength, char **ascii) {
  int   r;
  int   w;
  int   length;
  unsigned char *ascii7bit;

  /* Allocate sufficient memory for the 7-bit string */
  ascii7bit=malloc(pdulength*2+1);
  w=0;

  for (r=0; r<pdulength; r++) {
    if (r%7==0) {
      ascii7bit[w++]=(pdu[r]<<0)&0x7F;
    } else if (r%7==6) {
      ascii7bit[w++]=((pdu[r]<<6)|(pdu[r-1]>>2))&0x7F;
      ascii7bit[w++]=(pdu[r]>>1)&0x7F;
    } else {
      ascii7bit[w++]=((pdu[r]<<(r%7))|(pdu[r-1]>>(7+1-(r%7))))&0x7F;
    }
  }

  length=convert_7bit_to_ascii(ascii7bit,w,ascii);
  free(ascii7bit);

  return length;
}




/*
 *  Convert an ISO-8859-1 ASCII string to an array of PDU-coded bytes
 *
 *  *ascii  The ISO-cleartext to convert
 *
 *  **pdu   Pointer to an array where the pdu-bytes will be stored in, this
 *          function allocates the neccesary memory for the array
 *
 *  Returns the number of bytes stored in the pdu-array
 *
 *  Note:  Don't forget to free() the pdu-array when you're finished with it.
 *
```

```
 */

int ascii_to_pdu(char *ascii, unsigned char **pdu) {
   int r;
   int w;
   char *ascii7bit;
   int  len7bit;

   /* Start by converting the ISO-string to a 7bit-string */
   len7bit=convert_ascii_to_7bit(ascii,&ascii7bit);
   *pdu=malloc(len7bit);

   /* Now, we can create a PDU string by packing the 7bit-string */
   r=0;
   w=0;
   while (r<len7bit) {
      (*pdu)[w]=((ascii7bit[r]>>(w%7))&0x7F) | ((ascii7bit[r+1]<<(7-(w%7)))&0xFF);
      if ((w%7)==6) r++;
      r++;
      w++;
   }

   free(ascii7bit);
   return w;
}
```

5.4. Ficheiro decodPDU.c

```
#include<stdio.h>
#include<string.h>
#include<conio.h>
#include <malloc.h>
#include "pduconv.cpp"

#define chartohex(a) ((a<'A')? (a-'0'):(a-'A'+10))
/*
unsigned char chartohex(char a)
{ char r;
  if (a<'A') return(a-'0');
  else return(a-'A'+10);
}
*/
//-------------------------------------------------------------------------------------------

char *decodPDU(char sms[])
//void main(void)
{

      int i,n;
//    char sms[256]="0011000B916407281553F80000AA0AE8329BFD4697D9EC37";
//    char sms[256]="07917238010010F5040BC87238880900F10000993092516195800 3C16010";
      int len_centro_msg,len_origem_msg;

      unsigned char PDU_type;
      char *ptr;
```

```
        char _7bits;
        char bin_sms[141];
        char *ascii;

//------ o necess rio para usar decode_pdu() ------!

/*#include "sms_serv.h"
#include "gsmdev.h"
#include "pdu.h"
#define debug 1

    struct mbox_item message;
    cap_matrix capmatrix;
    struct gsms_def gsm;
    int default_alphabet = DEFAULT_ALPHABET;
//-------

        printf("Mensagem a descodificar:\n%s\n",sms);
        decode_pdu(sms, &message, gsm.capmatrix, default_alphabet);

        getch();
        */

        ptr=sms;
        printf("\nMensagem a descodificar:\n%s\n",sms);
        len_centro_msg= chartohex(sms[0])<<4 | chartohex(sms[1]);
        printf("SMSC address len = %d\n",len_centro_msg);

        printf("Tipo de numero 0x%c%c\n",sms[2],sms[3]);

        printf("SMSC Address = ");
                for(i=4;i<(len_centro_msg*2+2-2);i=i+2){
                        printf("%c",sms[i+1]);
                        printf("%c",sms[i]);
                }
                printf("%c",sms[i+1]);
                if (sms[i]!='F') printf("%c",sms[i]);
                printf("\n");

    printf("PDU type =0x%c%c\n",sms[((len_centro_msg)*2)+2],sms[((len_centro_msg)*2)+3]);

    PDU_type=chartohex(sms[len_centro_msg*2+2])<<4 | chartohex(sms[len_centro_msg*2+3]);

                switch(PDU_type & 0x03){
                case 0x00: printf("SMS message type = SMS-DELIVER ou SMS-DELIVER
REPORT\n");
                        break;
                case 0x01: printf("SMS message type = SMS-SUBMIT\n");
                        break;
                case 0x02: printf("SMS message type = SMS-STATUS REPORT\n");
                        break;
                case 0x03: printf("SMS message type = SMS-COMMAND\n");
                };

                if ((PDU_type & 0x03)!=0)
                {
```

```
                //SMS-SUBMIT
                printf("MR 0x%c%c\n",sms[((len_centro_msg)*2)+4],sms[((len_centro_msg)*2)+5]);
                printf("Sender address len
=%c%c\n",sms[((len_centro_msg)*2)+6],sms[((len_centro_msg)*2)+7]);
                len_origem_msg=chartohex(sms[len_centro_msg*2+6])<<4 |
chartohex(sms[len_centro_msg*2+7]);

                printf("Tipo de numero
0x%c%c\n",sms[((len_centro_msg)*2)+8],sms[((len_centro_msg)*2)+9]);

                printf("Numero origem : ");
                for(i=len_centro_msg*2+10;i<(len_centro_msg*2)+10+len_origem_msg-2;i=i+2){
                        printf("%c",sms[i+1]);
                        printf("%c",sms[i]);
                }
                printf("%c",sms[i+1]);
                if (len_origem_msg%2==0) printf("%c",sms[i]);
                printf("\n");i=i+2;

                printf("PID 0x%c%c\n",sms[i],sms[i+1]);i=i+2;

                printf("DCS 0x%c%c\n",sms[i],sms[i+1]);
                _7bits=0;
                if (sms[i]=='0') {printf("Default Alphabet ->7bits data coding\n");_7bits=1;}
                else if ((sms[i]=='F') && (chartohex(sms[i+1]) & 0x04)) printf("8bits data coding\n");
                else printf("DCS inv lido\n");
                i=i+2;

                //VPF -->validity period format
                if ((PDU_type & 0x18) == 0x10) // VP relative (1 byte)
                    {   printf("VP 0x%c%c\n",sms[i],sms[i+1]);i=i+2;}
                else if ((PDU_type & 0x18) == 0x18) // VP absolute (7 byte)
                    { printf("VP 0x");
                        for(n=i;n<i+7*2;n++) printf("%c",sms[n]);
                        i=n; printf("\n");
                    }
                   else printf("VP not present\n");
                printf("UDL 0x%c%c\n",sms[i],sms[i+1]);i=i+2;
                ptr=sms+i;
                printf("HEX msg : %s\n",ptr);

                if (_7bits) {
                  for (n=0;n<strlen(ptr);n=n+2) bin_sms[n/2]=chartohex(ptr[n])<<4 |
chartohex(ptr[n+1]);
                  pdu_to_ascii((unsigned char *)bin_sms, strlen(ptr)/2, &ascii);
                  printf("MSG : %s\n",ascii);
                  free(ascii);
                  }

                }//if

                else if ((PDU_type & 0x03)==0)
                {
                printf("Sender address len
=0x%c%c\n",sms[((len_centro_msg)*2)+4],sms[((len_centro_msg)*2)+5]);
```

```
                len_origem_msg=chartohex(sms[len_centro_msg*2+4])<<4 |
chartohex(sms[len_centro_msg*2+5]);

                printf("Tipo de numero
0x%c%c\n",sms[((len_centro_msg)*2)+6],sms[((len_centro_msg)*2)+7]);

                printf("Numero origem : ");

                for(i=len_centro_msg*2+8;i<(len_centro_msg*2)+8+len_origem_msg-2;i=i+2){

                        printf("%c",sms[i+1]);
                        printf("%c",sms[i]);
                }
                printf("%c",sms[i+1]);
                if (len_origem_msg%2==0) printf("%c",sms[i]);
                printf("\n");i=i+2;

                printf("PID 0x%c%c\n",sms[i],sms[i+1]);i=i+2;
                printf("DCS 0x%c%c\n",sms[i],sms[i+1]);
                _7bits=0;
                if (sms[i]=='0') {printf("Default Alphabet ->7bits data coding\n");_7bits=1;}
                else if ((sms[i]=='F') && (chartohex(sms[i+1]) & 0x04)) printf("8bits data coding\n");
                else printf("DCS inv lido\n");
                i=i+2;


                printf("SCTS \n");


                for(n=i;n<i+7*2-2;n=n+2){

                        printf("%c",sms[n+1]);
                                if(n==i+4 || n==i+10)
                                        printf("%c  ",sms[n]);
                                else
                                        printf("%c-",sms[n]);
                }

                i=n+2;
                printf("\nUDL 0x%c%c\n",sms[i],sms[i+1]);i=i+2;
                ptr=sms+i;
                printf("HEX msg : %s\n",ptr);

                if (_7bits) {
                  for (n=0;n<strlen(ptr);n=n+2) bin_sms[n/2]=chartohex(ptr[n])<<4 |
chartohex(ptr[n+1]);
                  pdu_to_ascii((unsigned char *)bin_sms, strlen(ptr)/2, &ascii);
                  printf("MSG : %s\n",ascii);
                  return ascii;
                  free(ascii);
                  }


                }//else if
//-----------------------------------
```

```
}

void main(void)
{
      //deliver
      char
sms[256]="07915391260100002 40C9153913652867800004070404154 4080C4E8329BFD4697D9EC
77";
      //submit
//    char sms[256]="0011000C91539136528678000 0AA0AE8329BFD4697D9EC77";

      decodPDU(sms);
      getch();
}
```

5.5. Ficheiro tratar_tlm.cpp

```
#include "porta_serie.hpp"
#include<stdio.h>
#include<string.h>
//#include<stdlib.h>
#include<conio.h>
//#include<time.h>
#include<windows.h>
#include <malloc.h>
#include "pduconv.cpp"
#include "decodPDU.c"

COMPort aPort ("COM1");//para trocar a porta COM!

//-------------------------------------------------------------------------------------------
// funcao que envia comandos AT

void enviaAT(char sms[])
{
      int i=0;

// Para enviar caracteres

      while(sms[i]!=0)
      {
      aPort.write(sms[i]);
      i++;
      }
      printf("Enviou\n\n");

}
//-------------------------------------------------------------------------------------------
// funcao para receber as respostas do MT (telemóvel)
void recebeMT(char answer2[])
{
      int i=0,aux=0;

      for(i=0;i<700;i++){

                  answer2[i]=       aPort.read();
```

```
                if(answer2[i]=='\n'){
                        answer2[i+1]=0;

                        break;
                }
        }
}
//-------------------------------------------------------------------------------------------

char *decodPDU(char *sms);

//-------------------------------------------------------------------------------------------
//função para apagar mensagem da memória no indice indica da memória corrente
int apaga_sms(int indica)
{
        char sms[256];
        char sms2[256];
        int est1;

        sprintf(sms,"at+cmgd=%d\r",indica);
        enviaAT(sms);
        Sleep(100);
        est1=aPort.VerificaPorto();

        Sleep(100);
        if(est1!=0){
                                        do
                                        {
                                                recebeMT(sms2);
                                        }while(strncmp(sms2,"OK",2)!=0);
        }
        else
                return 4;
}

//-------------------------------------------------------------------------------------------
//função para receber mensagens do cartão SIM
char *recebeSMS(int conta,int *contar,int *contar2,char *buffer_msg)

                                        //conta-posição de memória a ler

                                        //contar-devolve nº de mensagens existentes nessa memória

                                        //buffer-guarda msg neste buffer
{
                                        //contar2-capacidade do cartão
        char resposta[512];
        int i=1,est1,aux=0,est2,aux1=0,aux2=0,aux3=0,aux4=0,aux5=0,aux6=0;
        char *resp;
        char resp2[512];
        char resp3[512];

                        sprintf(resp2,"at+cpms=\"SM\"\r");
```

```
                    enviaAT(resp2);
                    Sleep(100);
                    est2=aPort.VerificaPorto();
                    Sleep(100);

                    if(est2!=0)
                    {
                        do{
                            recebeMT(resposta);
                            if(strncmp(resposta,"+CPMS:",6)==0)
                                if(resposta[8]==','){
                                    aux1=resposta[7];//saber quantas sms
                                //existem nessa memória
                                    aux4=resposta[9]-0x30;
                                    aux5=resposta[10]-0x30;
                                    *contar=aux1-(0x30);
                                    aux6=(10*aux4)+aux5;
                                    *contar2=aux6;
                                }
                                else{
                                    aux1=resposta[7]-0x30;
                                    aux2=resposta[8]-0x30;
                                    aux3=(10*aux1)+aux2;
                                    aux4=resposta[10]-0x30;
                                    aux5=resposta[11]-0x30;
                                    aux6=(10*aux4)+aux5;
                                    *contar=aux3;
                                    *contar2=aux6;
                                }
                        }while(strncmp(resposta,"OK",2)!=0);

                    sprintf(resp3,"at+cmgr=%d\r",conta);
                    enviaAT(resp3);
                    Sleep(100);
                    est1=aPort.VerificaPorto();
                    Sleep(100);

                    if(est1!=0)
                        {
                        do{
                            recebeMT(resposta);
                            if(strncmp(resposta,"OK",2)!=0 &&
strncmp(resposta,"\r",1)!=0 && strncmp(resposta,"+CMGR:",6)!=0 &&
strncmp(resposta,"at+cmgr",7)!=0 && strncmp(resposta,"+CMS",4)!=0)
                            {
                                sprintf(resp3,"%s",resposta);
                                resp=decodPDU(resp3);
                                strcpy(buffer_msg,resp);
                                free(resp);
                                resp=buffer_msg;
                            }


                            if(strncmp(resposta,"+CMS",4)==0)
                            {
                                resp="OK";
```

```
                                        return resp;
                                }

                        }while(strncmp(resposta,"079",3)!=0 &&
strncmp(resposta,"+CMS",4)!=0);

                        do
                        {
                                recebeMT(resposta);
                        }while(strncmp(resposta,"OK",2)!=0);

                        return resp;
                }//if
        }//fecha 1º if

    else
    {
        resp="NULL";
        return resp;
    }

}
//---------------------------------------------------------------------
//função para receber msgs da memória do tlm
char *recebeSMS2(int conta1,int *contar1)
{

    char resposta[512];
    int i=1,est1,aux=0,aux1=0,aux2=0,aux3=0;
    char *resp;
    char resp2[512];
    char resp3[512];


    //ler mensagens da memória do tlm

    sprintf(resp2,"at+cpms=\"ME\"\r");
    enviaAT(resp2);
    Sleep(100);

    est1=aPort.VerificaPorto();

    Sleep(100);

    if(est1!=0)
    {
        do
        {
            recebeMT(resposta);
            if(strncmp(resposta,"+CPMS:",6)==0)
                if(resposta[8]==','){
                                aux=resposta[7];//saber quantas sms estão
no telemovel
                                *contar1=aux-(0x30);
                        }
                        else{
```

```
                                                        aux1=resposta[7]-0x30;
                                                        aux2=resposta[8]-0x30;
                                                        aux3=(10*aux1)+aux2;
                                                        *contar1=aux3;
                                                }
                }while(strncmp(resposta,"OK",2)!=0);


        }//fecha if

        else{

                resp="NULL";
                return resp;
        }

        //next


         //ler as mensagens da memória do tlm

                sprintf(resp3,"at+cmgr=%d\r",conta1);
                enviaAT(resp3);


                Sleep(100);
                est1=aPort.VerificaPorto();

                Sleep(100);


                if(est1!=0)
                {
                        do
                        {
                        recebeMT(resposta);
                                if(strncmp(resposta,"OK",2)!=0 && strncmp(resposta,"\r",1)!=0 &&
strncmp(resposta,"+CMGR:",6)!=0 && strncmp(resposta,"at+cmgr",7)!=0)
                                {
                                        sprintf(resp3,"%s",resposta);
                                        resp=decodPDU(resp3);

                                }


                                if(strncmp(resposta,"+CMS",4)==0)
                                {
                                        resp="OK";
                                        return resp;
                                }

                                }while(strncmp(resposta,"079",3)!=0 &&
strncmp(resposta,"+CMS",4)!=0);
                                //receber o ultimo OK
                                do
```

```
                                        {
                                                recebeMT(resposta);
                                        }while(strncmp(resposta,"OK",2)!=0);

                return resp;
                }//if
        }

//-------------------------------------------------------------------------------------
//funçao para enviar a mensagem
int enviaSMS (char *ascii1,char *numero)

{

    aPort.setBitRate (COMPort::br9600);
    aPort.setParity (COMPort::None);
    aPort.setDataBits (COMPort::db8);
    aPort.setStopBits (COMPort::sb1);
    aPort.setBlockingMode(10,10,10,100,1);

    int i=0,modo=0,n=0,j=0;
    int tam_sms3=0;
    int pdulen,asciilen,numlen,est,verifica=0;

    char answer2[512];
    char sms[512]={'0','0','1','1','0','0','0','C','9','1'};
    char num2[512];//string para nº de telemóvel
    char sms2[256]={'0','0','0','0','A','A'};
    char sms5[256];
    char    *ascii2;
     unsigned char *pdu;
    unsigned char pdu1[512];

     //-(1)----converte a sms de ascii para PDU---------
   asciilen=strlen(ascii1);

    pdulen=ascii_to_pdu(ascii1,&pdu);

   pdu_to_ascii(pdu,pdulen,&ascii2);//retorna o PDU para uma sms legivel

    for (i=0; i<pdulen;i++) {
  sprintf((char*)&pdu1[j],"%02hhX ",pdu[i]); //converte para string
     j=j+2;
                }

   // Free the allocated strings
   free(pdu);
   free(ascii2);
     //-(1)-----------------------------------------------

     //-(2)----prepara a sms a enviar--------------------


     // protecção para ter que se inserir o 351 seguido de 93,96,91
```

```
                numlen=strlen(numero);
                //troca os digitos do n de tlmvl 2 a 2
                for(i=0;i<numlen;i=i+2){

                        num2[i]=numero[i+1];
                        num2[i+1]=numero[i];
                        }
        //cola o n de tlmvl ao inicio(sms)
        for(i=10,j=0;i<10+numlen;i++,j++)
                sms[i]=num2[j];

        //cola TP-PID,TP-DCS e Periodo de validade (sms2) a seguir ao nº de tlmvl
        for(i=10+numlen,j=0;i<10+numlen+6;i++,j++)
                sms[i]=sms2[j];

        sprintf((char *)&sms[28],"%02hhX",asciilen);//tamanho da mensagem a enviar

        for(i=30,j=0;i<30+pdulen*2;i++,j++)
                sms[i]=pdu1[j];

        //-fim(2)-------------------------------------------------

        //-(3)---envio da sms via rs232-----------------------
        enviaAT("AT+CMGF=0\r");//coloca em modo PDU

        Sleep(100);
        est=aPort.VerificaPorto();

        Sleep(100);
                if(est!=0)
                        {
                        do{
                            recebeMT(answer2); //eco
                        }while(strncmp(answer2,"OK",2)!=0)

        sprintf((char*)sms5,"AT+CMGS=%d\r",(30+pdulen*2-2)/2);//at+cmgw escreve na memória do
    //tlm

        //at+cmgs envia para o tlm
        enviaAT((char*)sms5);
        Sleep(10);

        est=aPort.VerificaPorto();
                        }
                        else{
                                verifica=1;
                                return verifica;
                        }
        if(est!=0){ //abre if
                Sleep(100);
                recebeMT(answer2); //recebe o at+cmgw

        //string final para enviar ao tlm
        sms[30+pdulen*2]=26;

        for(i=0;i<=30+pdulen*2;i++)
```

```
        {
        aPort.write(sms[i]);
                }

        do{
                recebeMT(answer2); //eco

        }while(strncmp(answer2,"OK",2)!=0);

        }//fecha if
        else{
                                        verifica=1;
                                        return verifica;
                        }
}
```

5.6. Ficheiro api_win.cpp

```
#include "porta_serie.hpp"
#include <windows.h>
#include "resource.h"
#include "stdio.h"
#include <malloc.h>

const char ClassName[] = "MainWindowClass";
const char DialogClassName[] = "DialogClass";

HWND hWndButton;
HWND hWndEditBox;
HWND hWndEditBox1;
HWND hWndEditBox2;
HWND hWndEditBox3;
HWND hWndStatic;
HWND hDlgButton;
HWND hWndDlgBox;

#define IDC_DLG_TEXT 234
int n_msg=1;
int n_msg2=1;
int aux=0;
int aux1=0;
int indice1=1;
int indice2=1;
int escolha=0;

//-----------------------------------------------
extern int enviaSMS(char *texto,char *numero);
extern char *recebeSMS2(int contador,int *verifica);
extern char *recebeSMS(int contador1,int *verifica1,int *verifica2,char *buffer);
extern int Porta();
extern int apaga_sms(int indicador);
//-----------------------------------------------

LRESULT CALLBACK DlgProc( HWND   hWndDlg,UINT   Msg,WPARAM wParam,LPARAM
lParam )
{
```

```
  int verificador1=0;
switch (Msg)
{
   case WM_CREATE:
   {
                        //botao sim
      hDlgButton = CreateWindowEx(
      0,
      "BUTTON",
      "Sim",
      WS_VISIBLE | WS_CHILD | BS_DEFPUSHBUTTON|WS_TABSTOP,
      30,
      20,
      80,
      20,
      hWndDlg,
      (HMENU)IDC_SIM,
      (HINSTANCE)GetWindowLong(hWndDlg, GWL_HINSTANCE),
      NULL);

      if (!hDlgButton)
         MessageBox(NULL, "Dialog Button Failed.", "Error", MB_OK | MB_ICONERROR);

                        //botao nao
                        hDlgButton = CreateWindowEx(
      0,
      "BUTTON",
      "Não",
      WS_VISIBLE | WS_CHILD | BS_DEFPUSHBUTTON|WS_TABSTOP,
      130,
      20,
      80,
      20,
      hWndDlg,
      (HMENU)IDC_NAO,
      (HINSTANCE)GetWindowLong(hWndDlg, GWL_HINSTANCE),
      NULL);

      if (!hDlgButton)
         MessageBox(NULL, "Dialog Button Failed.", "Error", MB_OK | MB_ICONERROR);

    //  return TRUE;
   }
   break;

   case WM_COMMAND: // comando da caixa de botões SIM/NÃO
   {
      switch(LOWORD(wParam))
      {
         case IDC_SIM:
         {
            switch(HIWORD(wParam))
            {
               case BN_CLICKED:

               if(escolha==1){//apagar sms do cartão SIM
```

```
                verificador1=apaga_sms(n_msg-1);
                indice1=2;
                SetWindowText(hWndEditBox1,"");
                        }

                if(escolha==2)//apagar sms do telemóvel
                {
                verificador1=apaga_sms(n_msg2-1);
                indice2=2;
                SetWindowText(hWndEditBox2,"");
                                }

                if(verificador1==4)//verificação da porta série
                MessageBox(NULL,"Ligue telemóvel","erro", MB_OK |MB_ICONERROR);
                        else{


      MessageBox(NULL, "Sms apagada.", "", MB_OK);
     SendMessage(hWndDlg, WM_CLOSE, 0, 0);
                        }
                  break;
                  }
            }
            break;
     case IDC_NAO:
            {
               switch(HIWORD(wParam))
               {
                 case BN_CLICKED:
                 SendMessage(hWndDlg, WM_CLOSE, 0, 0);

                  break;
               }
            }
            break;

         }
         return 0;
      }
      break;

      case WM_CLOSE:
         DestroyWindow(hWndDlg);
         hWndDlgBox = NULL;
      break;

      default:
         return (DefWindowProc(hWndDlg, Msg, wParam, lParam));
   }
}

//--------------------------------------------------------------------------------
LRESULT CALLBACK WndProc( HWND hWnd,UINT Msg,WPARAM wParam,LPARAM lParam
)
{
     DWORD TxtLen; //tamanho do buffer
```

```
    DWORD NumLen; //tamanho do numero
    char *pszMem; //buffer
    char *Num;
    int contador=0,i=0,verificador=0,verificacao=2;
    int variavel;
    int variavel1;
    int variavel2;

  //LPTSTR convBuff;

    char *answer2;
    char *answer3;
    char answer4[256];

  switch (Msg)
  {
     case WM_CREATE:
     {
        WNDCLASSEX  wc;
        wc.cbSize          = sizeof(WNDCLASSEX);
        wc.style           = 0;
        wc.lpfnWndProc     = (WNDPROC)DlgProc;
        wc.cbClsExtra      = 0;
        wc.cbWndExtra      = 0;
        wc.hInstance       = (HINSTANCE)GetWindowLong(hWnd, GWL_HINSTANCE);
        wc.hIcon           = NULL;
        wc.hIconSm         = NULL;
        wc.hCursor         = LoadCursor(NULL, IDC_ARROW);
        wc.hbrBackground   = (HBRUSH)(COLOR_WINDOW + 11);
        wc.lpszMenuName    = NULL;
        wc.lpszClassName   = DialogClassName;

        if(!RegisterClassEx(&wc))
        {
           MessageBox(NULL, "Failed To Register The Dialog Class.", "Error", MB_OK |
MB_ICONERROR);
           return FALSE;
        }

        //texto "Nº de destino:"
        hWndStatic = CreateWindow( "STATIC", NULL,
        WS_VISIBLE | WS_CHILD,
        10,
        15,
        200,
        20,
        hWnd,
        (HMENU)IDC_STATIC_TEXT,
        (HINSTANCE)GetWindowLong(hWnd, GWL_HINSTANCE),
        NULL);
        if (!hWndStatic) MessageBox(NULL, "Static Failed.", "Error", MB_OK | MB_ICONERROR);
        if (!SetWindowText(hWndStatic, "Nº de destino(351):")) MessageBox(NULL, "Failed To Set
    Text.", "Error", MB_OK | MB_ICONERROR);

        //texto "Mensagem a Enviar:"
        hWndStatic = CreateWindow( "STATIC", NULL,
```

```
        WS_VISIBLE | WS_CHILD,
        10,
        65,
        140,
        20,
        hWnd,
        (HMENU)IDC_STATIC_TEXT,
        (HINSTANCE)GetWindowLong(hWnd, GWL_HINSTANCE),
        NULL);
        if (!hWndStatic) MessageBox(NULL, "Static Failed.", "Error", MB_OK | MB_ICONERROR);
        if (!SetWindowText(hWndStatic, "Mensagem a Enviar:")) MessageBox(NULL, "Failed To Set
    Text.", "Error", MB_OK | MB_ICONERROR);

        //texto "Mensagens existentes no cartão SIM:"
        hWndStatic = CreateWindow(
        "STATIC",
        NULL,
        WS_VISIBLE | WS_CHILD,
        10,
        192,
        250,
        20,
        hWnd,
        (HMENU)IDC_STATIC_TEXT2,
        (HINSTANCE)GetWindowLong(hWnd, GWL_HINSTANCE),
        NULL);
        if (!hWndStatic)MessageBox(NULL, "Static Failed.", "Error", MB_OK | MB_ICONERROR);
        if (!SetWindowText(hWndStatic, "Mensagens existentes no cartão
SIM:"))MessageBox(NULL, "Failed To Set Text.", "Error", MB_OK | MB_ICONERROR);


        //texto "Mensagens existentes na memória do tlm:"
        hWndStatic = CreateWindow(
        "STATIC",
        NULL,
        WS_VISIBLE | WS_CHILD,
        10,
        318,
        250,
        20,
        hWnd,
        (HMENU)IDC_STATIC_TEXT2,
        (HINSTANCE)GetWindowLong(hWnd, GWL_HINSTANCE),
        NULL);
        if (!hWndStatic)MessageBox(NULL, "Static Failed.", "Error", MB_OK | MB_ICONERROR);
         if (!SetWindowText(hWndStatic, "Mensagens existentes no telemóvel:"))MessageBox(NULL,
    "Failed To Set Text.", "Error", MB_OK | MB_ICONERROR);



    //edit de nº de destino

        hWndEditBox3 = CreateWindow(
        "EDIT",
        "",
```

```
WS_CHILD | WS_VISIBLE |WS_BORDER,
10,
30,
250,
20,
hWnd,
(HMENU)IDC_EDITBOX_TEXT3,
(HINSTANCE)GetWindowLong(hWnd, GWL_HINSTANCE),
NULL);
 if (!hWndEditBox3)
    MessageBox(NULL, "Edit Box Failed.", "Error", MB_OK | MB_ICONERROR);



  //edit da sms a enviar
 hWndEditBox = CreateWindow(
"EDIT",
"",
WS_CHILD | WS_VISIBLE  |WS_BORDER| ES_MULTILINE ,
10,
82,
250,
100,
hWnd,
(HMENU)IDC_EDITBOX_TEXT,
(HINSTANCE)GetWindowLong(hWnd, GWL_HINSTANCE),
NULL);
 if (!hWndEditBox)
    MessageBox(NULL, "Edit Box Failed.", "Error", MB_OK | MB_ICONERROR);

//edit da sms recebida no cartão SIM
hWndEditBox1 = CreateWindow(
"EDIT",
"",
WS_CHILD | WS_VISIBLE  |WS_BORDER| ES_MULTILINE | ES_AUTOVSCROLL ,
10,
210,
250,
100,
hWnd,
(HMENU)IDC_EDITBOX_TEXT2,
(HINSTANCE)GetWindowLong(hWnd, GWL_HINSTANCE),
NULL);
if (!hWndEditBox1)
    MessageBox(NULL, "Edit Box Failed.", "Error", MB_OK | MB_ICONERROR);


//edit da sms recebida na memória do tlm
hWndEditBox2 = CreateWindow(
"EDIT",
"",
WS_CHILD | WS_VISIBLE  |WS_BORDER| ES_MULTILINE | ES_AUTOVSCROLL ,
10,
335,
250,
```

```
            100,
            hWnd,
            (HMENU)IDC_EDITBOX_TEXT3,
            (HINSTANCE)GetWindowLong(hWnd, GWL_HINSTANCE),
            NULL);
            if (!hWndEditBox2)
               MessageBox(NULL, "Edit Box Failed.", "Error", MB_OK | MB_ICONERROR);
                           HINSTANCE hInstance = (HINSTANCE)GetWindowLong(hWnd,
GWL_HINSTANCE);




           //botão apagar nº de destino
            hWndButton = CreateWindowEx(
            0,
            "BUTTON",
            "Apagar",//texto do botao
            WS_VISIBLE | WS_CHILD,
            300,
            30,
            90,
            20,
            hWnd,
            (HMENU) BOTAO_APAGAR3,
            hInstance,
            NULL);


              //botao enviar
            hWndButton = CreateWindowEx(
            0,
            "BUTTON",
            "Enviar",//texto do botao
            WS_VISIBLE | WS_CHILD,
            300,
            90,
            90,
            20,
            hWnd,
            (HMENU) BOTAO_ENVIAR,
            hInstance,
            NULL);

            //botao apagar sms enviada
            hWndButton = CreateWindowEx(
            0,
            "BUTTON",
            "Apagar",//texto do botao
            WS_VISIBLE | WS_CHILD,
            300,
            120,
            90,
            20,
```

```
        hWnd,
        (HMENU) BOTAO_APAGAR,
        hInstance,
        NULL);

    //botao receber mensagens do cartão SIM

        hWndButton = CreateWindowEx(
        0,
        "BUTTON",
        "Receber",//texto do botao
        WS_VISIBLE | WS_CHILD,
        300,
        210,
        90,
        20,
        hWnd,
        (HMENU) BOTAO_RECEBER,
        hInstance,
        NULL);

       //botão apagar sms recebida do cartão SIM
        hWndButton = CreateWindowEx(
        0,
        "BUTTON",
        "Apagar",//texto do botao
        WS_VISIBLE | WS_CHILD,
        300,
        240,
        90,
        20,
        hWnd,
        (HMENU) BOTAO_APAGAR2,
        hInstance,
        NULL);

        //botao apagar texto da sms recebida do SIM
        hWndButton = CreateWindowEx(
        0,
        "BUTTON",
        "Apagar texto",//texto do botao
        WS_VISIBLE | WS_CHILD,
        300,
        270,
        90,
        20,
        hWnd,
        (HMENU) BOTAO_APAGAR5,
        hInstance,
        NULL);


       //botao receber mensagens do telemóvel

        hWndButton = CreateWindowEx(
        0,
```

```
    "BUTTON",
    "Receber",//texto do botao
    WS_VISIBLE | WS_CHILD,
    300,
    340,
    90,
    20,
    hWnd,
    (HMENU) BOTAO_RECEBER2,
    hInstance,
    NULL);

   //botão apagar sms recebida do tlm
    hWndButton = CreateWindowEx(
    0,
    "BUTTON",
    "Apagar",//texto do botao
    WS_VISIBLE | WS_CHILD,
    300,
    370,
    90,
    20,
    hWnd,
    (HMENU) BOTAO_APAGAR4,
    hInstance,
    NULL);

    //botao apagar texto da sms recebida do telemovel
    hWndButton = CreateWindowEx(
    0,
    "BUTTON",
    "Apagar texto",//texto do botao
    WS_VISIBLE | WS_CHILD,
    300,
    400,
    90,
    20,
    hWnd,
    (HMENU) BOTAO_APAGAR6,
    hInstance,
    NULL);


}
break;

case WM_COMMAND:
{
   switch(LOWORD(wParam))
   {

                case BOTAO_RECEBER://cartão SIM

                                escolha=1;
                                // ler msg do cartão SIM
```

```
answer3=recebeSMS(n_msg,&variavel,&variavel2,answer4);

//protecção da porta série
if(answer3=="NULL"){
MessageBox(NULL,"Ligue telemóvel","erro", MB_OK |MB_ICONERROR);
        break;
}
else
if(answer3=="OK")//quando não existe mensagem nessa posição de memória
        {
                do
                {
                        n_msg=n_msg+1;

answer3=recebeSMS(n_msg,&variavel,&variavel2,answer4);

SendMessage(hWndEditBox1, WM_SETTEXT, 0,(LPARAM) answer4);
Sleep(100);
n_msg=n_msg+1;

if(indice1==2){
                indice1=1;
                variavel=variavel-1;
                        }
if(n_msg==variavel2+1)
        n_msg=1;

        }while(answer3=="OK");
}// fecha if
        else
        {
                Sleep(100);
                SendMessage(hWndEditBox1, WM_SETTEXT, 0,(LPARAM) answer4);
                Sleep(100);
                n_msg=n_msg+1;

                if(indice1==2){
                        indice1=1;
                        variavel=variavel-1;
                        }

                if(n_msg==variavel2+1)
                        n_msg=1;
                        }//fecha else
                        break;

case BOTAO_RECEBER2://Telemóvel

escolha=2;
answer2=recebeSMS2(n_msg2,&variavel1);


//protecção da porta série
if(answer2=="NULL")
        {
```

```
                MessageBox(NULL,"Ligue telemóvel","erro", MB_OK |MB_ICONERROR);
                        break;
}
        else
        if(answer2=="OK")//quando não existe mensagem nessa posição de memória
                {
                do
                        {
                //MessageBox(NULL,"Nao existe mensagem","erro", MB_OK |MB_ICONERROR);
                        Sleep(100);
                        n_msg2=n_msg2+1;
                        aux1=aux1+1;


answer2=recebeSMS2(n_msg2,&variavel1);
SendMessage(hWndEditBox2, WM_SETTEXT, 0,(LPARAM) answer2);

Sleep(100);
n_msg2=n_msg2+1;

if(indice2==2){
                indice2=1;
                variavel1=variavel-1;
                                        }

if(n_msg2==variavel1+1+aux1){
                        n_msg2=1;
                        }
}while(answer2=="OK");
                }// fecha if
                else
                {
                Sleep(100);
                SendMessage(hWndEditBox2, WM_SETTEXT, 0,(LPARAM) answer2);


        MessageBox(NULL,"mensagem","", MB_OK |MB_ICONINFORMATION);
                Sleep(100);
                n_msg2=(n_msg2+1);

                if(indice2==2){
                indice2=1;

        variavel1=variavel-1;
                }

                if(n_msg2==variavel1+1+aux1){
                                n_msg2=1;
                                        }
                                        }
                                break;

        case BOTAO_ENVIAR:

        NumLen = GetWindowTextLength(hWndEditBox3);
        TxtLen = GetWindowTextLength(hWndEditBox);
```

```
if(NumLen==0 && TxtLen==0)
MessageBox(NULL,"Insira numero e mensagem","ERRO", MB_OK |MB_ICONERROR);

else
if(NumLen==0 && TxtLen!=0)
MessageBox(NULL,"Insira numero","ERRO", MB_OK |MB_ICONERROR);
else

if(NumLen!=12)
MessageBox(NULL,"Numero incorrecto","ERRO", MB_OK |MB_ICONERROR);
else
if(TxtLen==0 && NumLen!=0 )
MessageBox(NULL,"Insira mensagem","ERRO", MB_OK |MB_ICONERROR);

else
if(TxtLen>160)
MessageBox(NULL,"O nº máximo de caracteres é 160","ERRO", MB_OK | MB_ICONERROR);


else

if(TxtLen && NumLen==12)
{
pszMem  = (char *) VirtualAlloc((LPVOID) NULL,(TxtLen + 1), MEM_COMMIT,
                                                        PAGE_READWRITE);
GetWindowText(hWndEditBox,pszMem,(int) TxtLen + 1);

Num  = (char *) VirtualAlloc((LPVOID) NULL,(NumLen + 1), MEM_COMMIT,
                                                        PAGE_READWRITE);
GetWindowText(hWndEditBox3,Num,(int) NumLen + 1);



//código  enviar sms aqui----------------------------------
    if(Num[0]!='3' || Num[1]!='5' || Num[2]!='1' || Num[3]!='9' || (Num[4]!='1'&& Num[4]!='3' &&
Num[4]!='6'))

MessageBox(NULL,"Insira 351 seguido de 91,93 ou 96 ","ERRO", MB_OK |MB_ICONERROR);

else{
            verificador=enviaSMS(pszMem,Num);

                    //protecção da porta série
                    if(verificador==1)
                    MessageBox(NULL,"Ligue telemóvel","ERRO", MB_OK |MB_ICONERROR);

                            else
                            {

MessageBox(NULL,"Mensagem enviada com sucesso","", MB_OK | MB_ICONINFORMATION);
                                    Sleep(1000);


            VirtualFree(pszMem, 0, MEM_RELEASE);
```

```
                VirtualFree(Num   , 0, MEM_RELEASE);
                                        }//fecha else
                                }

                        }

        break;

    case BOTAO_APAGAR3:
        SendMessage(hWnd, APAGA_NUMERO, 0, 0);
        break;

        case BOTAO_APAGAR:
        SendMessage(hWnd, APAGA_ENVIAR, 0, 0);
        break;

        case BOTAO_APAGAR5:
        SendMessage(hWnd, APAGA_RECEBER, 0, 0);
        break;

        case BOTAO_APAGAR6:
        SendMessage(hWnd, APAGA_RECEBER2, 0, 0);
        break;

         case BOTAO_APAGAR2:
              {
                 switch(HIWORD(wParam))
                 {
                    case BN_CLICKED:
                    {
                       if (!hWndDlgBox)
                       {
                          hWndDlgBox = CreateWindowEx(
                          WS_EX_DLGMODALFRAME | WS_EX_TOPMOST,
                          DialogClassName,
                          "Deseja realmente apagar a mensagem?",
                          WS_VISIBLE | WS_POPUP | WS_CAPTION|WS_TABSTOP,
                          250,
                           250,
                          250,
                          100,
                          hWnd,
                          NULL,
                          (HINSTANCE)GetWindowLong(hWnd, GWL_HINSTANCE),
                          NULL);

                          if (!hWndDlgBox)
                             MessageBox(NULL, "Dialog Box Failed.", "Error", MB_OK |
MB_ICONERROR);
                       }
                    }
                    break;
                 }
              }//fecha BOTAO_APAGAR2
              break;
```

```
                                    case BOTAO_APAGAR4:
            {
               switch(HIWORD(wParam))
               {
                  case BN_CLICKED:
                  {
                     if (!hWndDlgBox)
                     {
                        hWndDlgBox = CreateWindowEx(
                        WS_EX_DLGMODALFRAME | WS_EX_TOPMOST,
                        DialogClassName,
                        "Deseja realmente apagar a mensagem?",
                        WS_VISIBLE | WS_POPUP | WS_CAPTION|WS_TABSTOP,
                        250,
                                                                  250,
                        250,
                        100,
                        hWnd,
                        NULL,
                        (HINSTANCE)GetWindowLong(hWnd, GWL_HINSTANCE),
                        NULL);

                        if (!hWndDlgBox)
                           MessageBox(NULL, "Dialog Box Failed.", "Error", MB_OK |
MB_ICONERROR);
                     }
                  }
                  break;
               }
            }//fecha BOTAO_APAGAR2
            break;


         }//fecha LOWORD(wParam)

      } //fecha WM_COMMAND
      break;
//-----------------------------------------------------------------
      case APAGA_ENVIAR:
      SetWindowText(hWndEditBox,"");
               break;

      case APAGA_RECEBER:
      SetWindowText(hWndEditBox1,"");
               break;
      case APAGA_RECEBER2:
      SetWindowText(hWndEditBox2,"");
               break;



      case APAGA_NUMERO:
                        SetWindowText(hWndEditBox3,"");
      break;

//-----------------------------------------------------------------
```

```
        case WM_CLOSE:
           DestroyWindow(hWnd);
        break;

        case WM_DESTROY:
           PostQuitMessage(0);
        break;

        default:
           return (DefWindowProc(hWnd, Msg, wParam, lParam));
     }//fecha switch(Msg)

     return 0;
  }//fecha LRESULT
  //-------------------------------------------------------------------------------------------
  INT WINAPI WinMain( HINSTANCE  hInstance,
                      HINSTANCE  hPrevInstance,
                      LPSTR      lpCmdLine,
                      INT        nCmdShow )
  {
     WNDCLASSEX    wc;

     wc.cbSize          = sizeof(WNDCLASSEX);
     wc.style           = 0;
     wc.lpfnWndProc     = (WNDPROC)WndProc;
     wc.cbClsExtra      = 0;
     wc.cbWndExtra      = 0;
     wc.hInstance       = hInstance;
     wc.hIcon           = LoadIcon(hInstance, MAKEINTRESOURCE(IDI_ICON));
     wc.hIconSm         = LoadIcon(hInstance, MAKEINTRESOURCE(IDI_ICON));
     wc.hCursor         = LoadCursor(NULL, IDC_ARROW);
     wc.hbrBackground   = (HBRUSH)(COLOR_WINDOW + 11);
     wc.lpszMenuName    = MAKEINTRESOURCE(IDR_MENU);
     wc.lpszClassName   = ClassName;



     if (!RegisterClassEx(&wc))
     {
        MessageBox(NULL, "Failed To Register The Window Class.", "Error", MB_OK |
  MB_ICONERROR);
        return 0;
     }

     HWND    hWnd;

     hWnd = CreateWindowEx(
     WS_EX_APPWINDOW,
     ClassName,
     "Envio e Recepção de SMS",
     WS_OVERLAPPEDWINDOW,
     CW_USEDEFAULT,
     CW_USEDEFAULT,
     420,
     485,
     NULL,
```

```
    NULL,
    hInstance,
    NULL);

    if (!hWnd)
    {
       MessageBox(NULL, "Window Creation Failed.", "Error", MB_OK | MB_ICONERROR);
       return 0;
    }

    ShowWindow(hWnd, SW_SHOW);
    UpdateWindow(hWnd);

    MSG    Msg;

    while (GetMessage(&Msg, NULL, 0, 0))
    {
       TranslateMessage(&Msg);
       DispatchMessage(&Msg);
    }

    return Msg.wParam;
}
```

5.7. Ficheiro porta_serie.cpp

```
//===========================================================================
//  General component library for WIN32
//  Copyright (C) 2000, UAB BBDSoft ( http://www.bbdsoft.com/ )
//
// This material is provided "as is", with absolutely no warranty expressed
// or implied. Any use is at your own risk.
//
// Permission to use or copy this software for any purpose is hereby granted
// without fee, provided the above notices are retained on all copies.
// Permission to modify the code and to distribute modified code is granted,
// provided the above notices are retained, and a notice that the code was
// modified is included with the above copyright notice.
//
//  The author of this program may be contacted at developers@bbdsoft.com
//===========================================================================
#ifndef _COMPORT_
  #include "porta_serie.hpp"
#endif

#ifndef _WINDOWS_
  #define WIN32_LEAN_AND_MEAN
  #include <windows.h>
#endif

#ifndef _STDEXCEPT_
  #include <stdexcept>
#endif

using namespace std;
```

```
//---------------------------------------------------------------------------
COMPort::COMPort (  char *  portName )
  : theDCB (NULL)
{

thePortHandle = (unsigned ) CreateFile ( portName
                                , GENERIC_READ | GENERIC_WRITE
                                , 0
                                , NULL
                                , OPEN_EXISTING
                                , FILE_FLAG_NO_BUFFERING
                                , NULL
                                );
if (thePortHandle == HFILE_ERROR)
{
  throw runtime_error ("COMPort: failed to open.");
} // endif

theDCB = new char [sizeof(DCB)];
getState();
setBlockingMode(10,0,0,10,1);
//setHandshaking(); //linha original
setHandshaking(false);

} // end constructor

//---------------------------------------------------------------------------

void COMPort::closeport()
{
// close serial port device
if (CloseHandle ((HANDLE)thePortHandle) == FALSE )
{
  throw runtime_error ("COMPort: failed to close.");
} // endif
thePortHandle=NULL;

} // end destructor

//---------------------------------------------------------------------------
COMPort::~COMPort()
{

delete [] theDCB;

// close serial port device
if (thePortHandle!=NULL)

if (CloseHandle ((HANDLE)thePortHandle) == FALSE )
{
  throw runtime_error ("COMPort: failed to close.");
} // endif



} // end destructor
```

```
//---------------------------------------------------------------------------
void COMPort::getState () const
{

if (!GetCommState ( (HANDLE) thePortHandle
                , (LPDCB) theDCB
                )
  )
{

  throw runtime_error ("COMPort: could not retrieve serial port state.");
} // endif

} // end COMPort::getState () const


//---------------------------------------------------------------------------
COMPort& COMPort::setState ()
{

if (!SetCommState ( (HANDLE) thePortHandle
                , (LPDCB) theDCB
                )
  )
{
  throw runtime_error ("COMPort: could not modify serial port state.");
} // endif

return *this;
} // end COMPort::setState ()


//---------------------------------------------------------------------------
COMPort& COMPort::setBitRate ( unsigned long Param )//Baudrate de 9600
{

DCB & aDCB = *((LPDCB)theDCB);
aDCB.BaudRate = Param;

return setState();
} // end COMPort::setBitRate (..)


//---------------------------------------------------------------------------
unsigned long COMPort::bitRate() const
{

DCB & aDCB = *((LPDCB)theDCB);

return aDCB.BaudRate;
} // end COMPort::bitRate () const


//---------------------------------------------------------------------------
```

```
COMPort& COMPort::setLineCharacteristics( char * inConfig )
{

COMMTIMEOUTS aTimeout;
if ( !BuildCommDCBAndTimeouts ( inConfig
                        , (LPDCB)theDCB
                        , &aTimeout
                        )
  )
{
  throw runtime_error ("COMPort: could not set line characteristics.");
} // endif

if ( ! SetCommTimeouts ( (HANDLE(thePortHandle))
                    , &aTimeout
                    )
  )
{
  throw runtime_error ("COMPort: could not set line characteristics.");
} // endif

return setState();
}


//---------------------------------------------------------------------------
char COMPort::read ()
{

char buffer;
DWORD charsRead = 0;

do
{
  if (! ReadFile ( (HANDLE(thePortHandle))
                , &buffer
                , sizeof(char)
                , &charsRead
                , NULL
                )
    )
  {
   throw runtime_error ("COMPort: read failed.");
  } // endif

} while ( !charsRead );

return  buffer;
} // end COMPort::read()


//---------------------------------------------------------------------------
unsigned long COMPort::read ( void *inBuffer
                        , const unsigned long inCharsReq
                        )
{
```

```
DWORD charsRead = 0;
if ( !ReadFile ( (HANDLE(thePortHandle))
            , inBuffer
            , inCharsReq
            , &charsRead
            , NULL
            )
  )
{
  throw runtime_error ("COMPort: read failed.");
} // endif

return charsRead;
} // end COMPort::read (..)


//---------------------------------------------------------------------------
COMPort & COMPort::write ( const char inChar )
{

char buffer = inChar;
DWORD charsWritten = 0;

if ( !WriteFile ( (HANDLE(thePortHandle))
            , &buffer
            , sizeof(char)
            , &charsWritten
            , NULL
            )
  )
{
  throw runtime_error ("COMPort: write failed.");
} // endif

return  *this;
} // end COMPort::write (..)
//---------------------------------------------------------------------------
unsigned long COMPort::write ( const void *inBuffer
                     , const unsigned long inBufSize
                     )
{

DWORD charsWritten = 0;

if ( !WriteFile ( (HANDLE(thePortHandle))
            , inBuffer
            , inBufSize
            , &charsWritten
            , NULL
            )
  )
{
  throw runtime_error ("COMPort: write failed.");
} // endif
```

```
return  charsWritten;
} // end COMPort::write()


//--------------------------------------------------------------------------
COMPort& COMPort::setxONxOFF ( bool Param )
{

DCB & aDCB = *((LPDCB)theDCB);
aDCB.fOutX = Param ? 1 : 0;
aDCB.fInX = Param ? 1 : 0;

return setState();
} // end COMPort::setxONxOFF (..)


//--------------------------------------------------------------------------
bool COMPort::isxONxOFF () const
{

DCB & aDCB = *((LPDCB)theDCB);

return (aDCB.fOutX && aDCB.fInX);
} // end COMPort::isxONxOFF () const


//--------------------------------------------------------------------------
COMPort& COMPort::setBlockingMode ( unsigned long inReadInterval
                        , unsigned long inReadMultiplyer
                        , unsigned long inReadConstant
                                                        , unsigned long inWriteMultiplyer
                                                        , unsigned long inWriteConstant
                        )
{

COMMTIMEOUTS commTimeout;

if ( !GetCommTimeouts ( (HANDLE(thePortHandle))
                , &commTimeout
                )
  )
{
  throw runtime_error ("COMPort: failed to retrieve timeouts.");
} // endif

commTimeout.ReadIntervalTimeout = inReadInterval;

if ( inReadInterval==MAXDWORD )
{
  commTimeout.ReadTotalTimeoutMultiplier = 0;
  commTimeout.ReadTotalTimeoutConstant = 0;
}
else
{
  commTimeout.ReadTotalTimeoutMultiplier = inReadMultiplyer;
  commTimeout.ReadTotalTimeoutConstant = inReadConstant;
```

```
 // novo ----------------------------------------------------------
   commTimeout.WriteTotalTimeoutMultiplier =inWriteMultiplyer;
   commTimeout.WriteTotalTimeoutConstant =inWriteConstant;



}  // endifelse



if ( !SetCommTimeouts ( (HANDLE(thePortHandle))
                    , &commTimeout
                    )
   )
{
   throw runtime_error ("COMPort: failed to modify timeouts.");
} // endif

return *this;
} // end COMPort::setBlockingMode (..)


//---------------------------------------------------------------------------
COMPort & COMPort::setHandshaking ( bool inHandshaking )
{

DCB & aDCB = *((LPDCB)theDCB);
if (inHandshaking)
{
   aDCB.fOutxCtsFlow = TRUE;
   aDCB.fOutxDsrFlow = FALSE;
   aDCB.fRtsControl = RTS_CONTROL_HANDSHAKE;
}
else
{
   aDCB.fOutxCtsFlow = FALSE;
   aDCB.fOutxDsrFlow = FALSE;
   aDCB.fRtsControl = RTS_CONTROL_ENABLE;
} // endifelse

return setState();
} // end COMPort::setHandshaking (..)


//---------------------------------------------------------------------------
unsigned long COMPort::getMaximumBitRate() const
{

COMMPROP aProp;
if ( !GetCommProperties ( (HANDLE)thePortHandle
                     , &aProp )
   )
{
   throw runtime_error ("COMPort: failed to retrieve port properties.");
} // endif
```

```
return aProp.dwMaxBaud;
} // end COMPort::getMaximumBitRate () const


//---------------------------------------------------------------------------
COMPort::MSPack COMPort::getModemSignals() const
{

MSPack aPack;
// 1 bit - DTR, 2 - bit RTS                    (output signals)
// 4 bit - CTS, 5 bit - DSR, 6 bit - RI, 7 bit - DCD  (input signals)
if ( !GetCommModemStatus ( (HANDLE)thePortHandle
                    , (LPDWORD)&aPack )
  )
{
  throw runtime_error ("COMPort: failed to retrieve modem signals.");
} // endif

return aPack;
} // end COMPort::getModemSignals () const


//---------------------------------------------------------------------------
COMPort& COMPort::setParity ( Parity Param )
{

DCB & aDCB = *((LPDCB)theDCB);
aDCB.Parity = Param;

return setState();
} // end COMPort::setParity (..)


//---------------------------------------------------------------------------
COMPort& COMPort::setDataBits ( DataBits Param )
{

DCB & aDCB = *((LPDCB)theDCB);
aDCB.ByteSize = Param;

return setState();
} // end COMPort::setDataBits (..)


//---------------------------------------------------------------------------
COMPort& COMPort::setStopBits ( StopBits Param )
{

DCB & aDCB = *((LPDCB)theDCB);
aDCB.StopBits = Param;

return setState();
} // end COMPort::setStopBits (..)
```

```
//---------------------------------------------------------------------------
COMPort::Parity COMPort::parity () const
{

DCB & aDCB = *((LPDCB)theDCB);

return (COMPort::Parity)aDCB.Parity;
} // end COMPort::parity () const


//---------------------------------------------------------------------------
COMPort::DataBits COMPort::dataBits () const
{

DCB & aDCB = *((LPDCB)theDCB);

return (COMPort::DataBits)aDCB.ByteSize;
} // end COMPort::dataBits () const


//---------------------------------------------------------------------------
COMPort::StopBits COMPort::stopBits () const
{

DCB & aDCB = *((LPDCB)theDCB);

return (COMPort::StopBits)aDCB.StopBits;
} // end COMPort::stopBits () cosnt
//---------------------------------------------------------------------------

int COMPort::  EstadoPorto()
{
DWORD evento=0;
int j=5;

/*SetupComm(
  (HANDLE)thePortHandle,     // handle to communications device
  (DWORD)sizeof (char),  // size of input buffer
  (DWORD)sizeof(char)  // size of output buffer
);*/


/*GetCommMask(                          //não sei para que serve...
  (HANDLE)thePortHandle,        // handle to communications device
  (LPDWORD) &evento);          // pointer to variable to receive events*/


/*SetCommMask(                                  // espera que caracter seja recebido neste PC para
enviar o resto
  (HANDLE)thePortHandle,    // handle to communications device
  evento | EV_TXEMPTY | EV_RXCHAR);  // mask that identifies enabled events*/


WaitCommEvent(
  (HANDLE)thePortHandle,        // handle to communications device
  (LPDWORD) &evento,          // pointer to variable to receive event
```

```
  NULL);                                                    // pointer to overlapped structure


//Sleep(100);


return evento;
}

//---------------------------------------------------------------------------
// funcao para esperar  a entrada de um caracter para ler o buffer
int COMPort::  EsperaPorto()
{

DWORD evento1=0;
int j;

SetCommMask(
  (HANDLE)thePortHandle,    // handle to communications device
       EV_RXCHAR | evento1); // mask that identifies enabled events
//     |EV_RXFLAG |EV_RLSD | EV_TXEMPTY );// envia um caracter de cada vez

WaitCommEvent(                                              // espera que caracter seja recebido neste PC
para enviar o resto
  (HANDLE)thePortHandle,        // handle to communications device
  (LPDWORD) &evento1,           // pointer to variable to receive event
  NULL);                                                    // pointer to overlapped structure


if(( evento1 & EV_TXEMPTY)!=0){
       j=1;
  //  printf("Event:%d\n",j);
       return j ;

}else
{
                j=0;
       //       printf("Event:%d\n",j);
                return j;
}

}
//---------------------------------------------------------------------------
// funcao para esperar que todos os caracteres sejam enviados
int COMPort::  EsperaPorto1()
{

       int i=0,aux=0;
       char answer2[256];

DWORD dwRead;
DWORD dwRes;
BOOL fWaitingOnRead = TRUE;
OVERLAPPED osReader = {0};
```

```
// Create the overlapped event. Must be closed before exiting
// to avoid a handle leak.
osReader.hEvent = CreateEvent(NULL, TRUE, FALSE, NULL);
SetCommMask(
  (HANDLE)thePortHandle,    // handle to communications device
       EV_RXCHAR ); // mask that identifies enabled events
//     |EV_RXFLAG |EV_RLSD | EV_TXEMPTY );// envia um caracter de cada vez


   dwRes = WaitForSingleObject(osReader.hEvent, 5000);
   switch(dwRes)
   {
     // Read completed.
     case WAIT_OBJECT_0:
        if (!GetOverlappedResult((HANDLE)thePortHandle, &osReader, &dwRead, FALSE))
          printf("Erro\n");
                          else{
          // Read completed successfully.
         answer2[i]=    read();
                          printf("%c",answer2[i]);

                          printf("Resposta:\n");

        //  Reset flag so that another opertion can be issued.

                  }
        break;

     case WAIT_TIMEOUT:
        // Operation isn't complete yet. fWaitingOnRead flag isn't
        // changed since I'll loop back around, and I don't want
        // to issue another read until the first one finishes.
        //
        // This is a good time to do some background work.
                  i=19;
                  printf("Timeout\n");

        break;

     default:
        // Error in the WaitForSingleObject; abort.
        // This indicates a problem with the OVERLAPPED structure's
        // event handle.
                          printf("default\n");
                  break;
   }
return 1;
}

//---------------------------------------------------------------------------
// função que verifica a existencia de dados no buffer
int COMPort::  VerificaPorto()
{
      LPDWORD lpErrors;
      LPCOMSTAT lpStat;
```

```
        COMSTAT ficha;
        lpStat=&ficha;


        ClearCommError(
                (HANDLE)thePortHandle,
                (LPDWORD) NULL,
                (LPCOMSTAT) lpStat);

        printf("Numero de bytes do buffer de entrada %d\n",ficha.cbInQue);

        return ficha.cbInQue;
}
```