

Universidad de Buenos Aires

Facultad De Ingeniería

Año 2011 - 2do Cuatrimestre

**Algoritmos Y Programación I (75.02)**

TRABAJO PRÁCTICO Nº 1

TEMA: Codificación ASCII / BASE64

FECHA: 2 de Noviembre de 2011

Correccion: Diciembre 2011

INTEGRANTES:

Sampayo, Sebastián - # 93793

<sebisampayo@gmail.com>

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

75.02 - Algoritmos y Programación I Ing. Martín Cardozo

**Introducción teórica**

El diseño del código Base64 se realizó teniendo en cuenta que el conjunto de caracteres que formarían parte del mismo fueran caracteres imprimibles y que estuvieran incluidos en la gran mayoría de los códigos de encriptación. La combinación de caracteres elegidos permite, por ejemplo, que la información no sea modificada al ser transportada por los sistemas de información de los servicios de e-mail.  
 La encriptación en base64 se basa en la representación de caracteres en forma binaria en bytes de 6 bits por cada carácter. Los caracteres que pertenecen al código base64 son los que van desde A-Z, a-z, 0-9 y los caracteres 62 y 63 que normalmente son ´/´ y ´+´ respectivamente, pero en algunas utilizaciones de base64 se modifican estos dos últimos caracteres. Por otro lado para lograr que siempre se puedan expresar todos los caracteres codificados en otros códigos de encriptación normalmente basados en caracteres que se almacenan en bytes de 8 bits se implementó un carácter que expresa los pad (´=´) que son necesarios para que a partir de un código que utilice caracteres de 8 bits se pueda traducir a código base64.  
 Lo anteriormente expresado se puede explicar de la siguiente forma: si uno tiene caracteres de 8bits y tiene que lograr agruparlos de forma tal que luego los pueda expresar como caracteres de 6bits entonces se deberán agrupar 3 caracteres de 8 bits de forma de obtener un conjunto de caracteres que utilizan 24 bits. A continuación, los 24 bits antes logrados a partir de 3 caracteres de 8 bits serán reagrupados en bytes de 6 bits (24 es el minimo común múltiplo de 6 y 8) obteniendo de esta forma la representación binaria de los caracteres en base64. En la siguiente tabla se expresa lo previamente explicado.

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **Text content** | **M** | | | | | | | | **a** | | | | | | | | | | **n** | | | | | | | | |
| **ASCII** | 77 | | | | | | | | 97 | | | | | | | | | | 110 | | | | | | | | |
| **Bit pattern** | 0 | 1 | 0 | 0 | 1 | 1 | 0 | 1 | 0 | 1 | 1 | 0 | 0 | | 0 | 0 | 1 | 0 | | 1 | 1 | | 0 | 1 | 1 | 1 | 0 |
| **Index** | 19 | | | | | | 22 | | | | | | | 5 | | | | | | | | 46 | | | | | |
| **Base64-encoded** | **T** | | | | | | **W** | | | | | | | **F** | | | | | | | | **u** | | | | | |

**Tabla 1:** en esta tabla se muestra la representación de caracteres ascii en binario y también su numero en la tabla de caracteres ascii, y a continuación se muestra el pasaje a código base64 y su representación en binario y el número del caracater en la tabla base64.

En el caso de que no sean ingresados tres caracteres ascii, se completan los 24bits con 0s y luego se corresponderá un ´=´ por cada carácter que fue completado con 0s.

El código base64 fue utilizado por primera vez por el protocolo  *Privacy-enhanced Electronic Mail (PEM),* luego *MIME* (multipurpose internet mail extensions) utilizó base64 basandose en la versión de PEM pero introdujo ciertos cambios como ser que la cantidad de caracteres por línea tenía que ser 76, excepto en la última que podían ser menos. Por otro lado indicó que los caracteres no pertenecientes a base 64 que se encontraran en medio del texto se omitirían en la decodificación. Más tarde, UTF-7 utilizó un sistema basado en base64 llamado Modified base64, tal modificación se trataba de la omisión del carácter ´=’ (UTF-7 se pretendió usar para las cabeceras de los correos (definido en [RFC 2047](http://tools.ietf.org/html/rfc2047)), y el carácter '=' se reserva en este contexto como carácter de escape para codificación [Quoted-Printable](http://es.wikipedia.org/wiki/Quoted-Printable" \o "Quoted-Printable)) en este caso se dejaban sin usar los últimos 4 digitos binarios del carácter si es que no se lograba agrupar los caracteres en grupos de 24 bits. Posteriormente, OpenPGP utilizó la codificación radix-64, también conocida como ASCII armor, la cual es muy similar a la codificación base64 utilizada por MIME salvo que se agrega una comprobación de redundancia cíclica de 24 bits.  
 La utilización de base64 y derivados fue expresada brevemente haciendo referencia a algunos de los muchos protocolos y especificaciones que utilizan base64 como código de encriptación.

**Procedimiento**

Implementación

Para la realización de un codificador/decodificador de texto en código ASCII a Base64, investigamos primero cuales eran las características y la forma en que opera cada uno de los códigos para poder determinar el mejor camino para codificar en ANSI C. A partir de la investigación realizada en la que observamos que podíamos optar por una filosofía al estilo UNIX en la que diseñaramos funciones con objetivos claros y simples, o bien podíamos organizarnos utilizando una filosofía menos ordenada en la que los distintos procesos estuvieran diseminados a lo largo de toda la función principal.  
 Si bien parece, por haberla calificado como clara y simple, que la filosofía UNIX es más sencilla de aplicar esto resulta a la inversa ya que requiere una gran planificación previo a la codificación y un gran nivel de detallismo en cuanto a las consideraciones, como ser excepciones, coherencia entre las distintas funciones, disposición y orden de jerarquía de las funciones. Mientras que en el caso de ir haciendo que los procesos se ejecuten a lo largo de toda la función principal tales inconvenientes de planificación detallada no se presentan.  
 Nosotros escogimos la filosofía al estilo unix ya que a pesar de que la planificación se tornara más engorrosa, los distintos procesos se realizarían en bloques bien definidos lo cual permitiría encontrar posibles errores o incluso mejorar posteriormente algún proceso en cuanto a rendimiento en rapidez o en almacenamiento.   
 Además, este modo de trabajo, posee el beneficio de producir pequeños módulos potencialmente útiles para otras aplicaciones futuras en las que se necesite realizar tareas ya codificadas en este programa. De esta forma, pensando a futuro, se ahorra tiempo de trabajo a largo plazo.   
 Sin embargo, esta filosofía no pudo ser adoptada para toda la modularización del programa, ya que ciertas tareas o bien requerían un acoplamiento por cohesión, o bien simplemente pensar el módulo de una tarea tan específica para aplicaciones futuras se tornaba demasiado complicado.

Dispusimos una serie de procesos para el caso de codificación (pasar de ASCII a Base64):

1. Validar los comandos por línea de argumentos
2. Seleccionar un modo de operación (encoder/decoder)
3. Leer 3 caracteres ASCII y guardarlos en un buffer1.
4. Codificar los caracteres ASCII a 4 caracteres en Base64 y guardarlos en un buffer2.
5. Imprimir los caracteres, en Base64.

También organizamos una lista de procesos para el caso de decodificar (pasar de Base64 a ASCII):

1. Validar los comandos por línea de argumentos
2. Seleccionar un modo de operación (encoder/decoder)
3. Leer 4 caracteres Base64 y guardarlos en un buffer1
4. Validar la entrada de texto, es decir comprobar que los caracteres leídos estén acorde con las especificaciones requeridas por código Base64.
5. Decodificar los caracteres Base64 a caracteres ASCII y guardarlos en un buffer2.
6. Imprimir los caracteres, en ASCII.

Como se puede ver los primeros dos procesos son compartidos en ambos casos y luego de acuerdo a lo que el usuario ingrese, se decide, si es válido lo ingresado, si se ingresa al modo encoder o decoder.

Lectura

En cuanto al proceso de lectura de caracteres se optó entre las siguientes opciones:

* Realizar un arreglo tan grande como caracteres se hubieran ingresado, determinando el número de caracteres ingresados a partir del operador sizeof() o a partir de una iteración en la que un contador frene cuando se encuentre EOF.
* Realizar un arreglo de 3 elementos y guardar en el de a tres caracteres.

La primera opción fue descartada porque requiere de un gran espacio en memoria y además podría ocurrir que el texto ingresado ocupe más lugar en memoria de lo que esta puede almacenar, o también puede ocurrir que si el número de caracteres que se ingresó es mayor al máximo número que puede adoptar un unsigned long, entonces se produzca un error. Además no es necesario.  
 La segunda opción fue la elegida ya que al procesar de esta forma la entrada, se obtienen grupos de caracteres de 24 bits, número que es el mínimo común múltiplo de 6 y 8 y se los almacena en un arreglo de 3 elementos. Por otro lado al procesar los caracteres se predijo que pudiera existir la situación en la que no se ingresen 3 caracteres sino ninguno, 1 o 2.

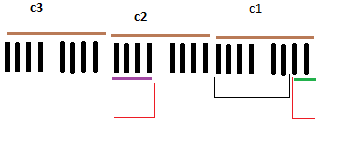
Linkeo

La lógica de lo que se presentará a continuación es utilizada en el modo decoder y encoder. La idea fue que debíamos linkear los caracteres que obteníamos de la lectura del stadard input (caracteres que eran leidos como ascii) con caracteres base 64. El carácter leído fue comparado con los caracteres del diccionario base 64 hasta encontrar coincidencia, luego sabiendo que el número del elemento es el número que corresponde con su denominación en base 64, es almacenado en un arreglo que luego será utilizado por la función decoder. En el caso de la utilización de esta misma lógica pero para el modo encoder, en la función de impresión de caracteres se realizó un linkeo entre caracteres base64 y ascii utilizando el arreglo que recibe de la función encoder y utilizándolo como el número de elemento del diccionario base64.

Codificación

En cuanto al proceso de codificación se idearon dos alternativas las cuales en un principio se pensó que solo diferían en cuanto al orden de la lectura de los caracteres que se encuentran en el arreglo, pero luego se descubrió que no era tan así. Las dos alternativas eran:

* Procesa el primer elemento del arreglo, es decir el primer carácter ingresado, y lo ingresa en un buffer inicializado a cero y de tipo unsigned int y se lo desplaza a izquierda 16 lugares. Luego se ingresa el segundo carácter al buffer y se lo desplaza 8 lugares. Por último, se ingresa el tercer carácter al buffer y ya se tiene completo el llenado de este buffer. A continuación se define una máscara (0x3F) y se hace un AND de bits con el buffer y almacenando tal resultado en un buffer2, en el último elemento. Luego siguiendo el mismo principio, se repetirá lo anterior pero desplazando primero el buffer hacia la derecha de a 6 lugares y almacenando lo obtenido llenando el buffer2 desde el último elemento hasta el primero en ese orden. Para las excepciones la lógica es la misma salvo que en caso de estar en la excepción 1 el desplazamiento a izquierda será 4 lugares y se realizara por única vez siguiendo luego con el AND de bits y un desplazamiento a derecha (6lugares) y otro AND de bits, llenando de esta forma los elementos 1 y 0 del buffer2 respectivamente; los elementos 2 y 3 se llenaran con ´=´. En el caso de la excepción 2 la lógica vuelve a ser parecida a la de la excepción 1.
* En esta alternativa los caracteres son ingresados al buffer siguiendo la misma lógica explicada anteriormente pero ingresando primero el último elemento del arreglo que contiene los caracteres leídos y siguiendo así hasta el primero. Desde aquí ya ni se presentara como se tendría que codificar ya que con la siguiente explicación se entenderá que esta forma es sumamente complicada. Viendo el diagrama siguiente, podemos ver que lo primero que se tiene que procesar es lo negro, luego se tiene que procesar lo rojo, teniendo que hacer una máscara para el caso negro y otra para el subcaso verde del caso rojo, desplazar lo obtenido y sumarlo con una máscara (desplazada hacia la derecha) para el subcaso violeta del caso rojo. Como se puede ver esto es muy complicado y ni siquiera se terminó con la decodificación de dos caracteres Base64.



**Conclusión**

A modo de conclusión, este grupo cree que el trabajo práctico ha sido un buen primer paso en diversas áreas de la programación. Estas áreas comprenden, el mundo de los protocolos, a través de la investigación que tuvo que realizarse en cuanto a la utilización de base64; la encriptación y compresión de datos en sistemas informáticos; la modularización de programas y sus criterios; la organización de código fuente en archivos separados por tareas (main, funciones, cabeceras); el trabajo en equipo y la coordinación mutua de un grupo de trabajo.

**Bibliografía**

* “Como programar en C/C++”, H.M. Deitel & P.J. Deitel, Segunda edición.
* “El lenguaje de programación C con base en ANSI C”, Kernighan & Ritchie, Segunda edición.
* Carpeta de clase.
* Ejemplos enviados por e-mail.
* <http://es.wikipedia.org/wiki/Base64>
* <http://en.wikipedia.org/wiki/Base64>