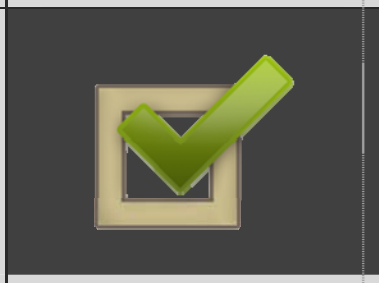
ALGORITMOS Y ESTRUCTURA DE DATOS



guía práctica interactiva MODULO 1.5

Nota para el alumno

Esta Guía Práctica Interactiva ha sido desarrollada por Pablo A. Sznajdleder para la cátedra de Algoritmos y Estructura de Datos de la Universidad Tecnológica Nacional (UTN.BA), República Argentina.

La presente no está totalmente depurada pudiendo contener errores de cualquier tipo: sintácticos, lógicos, de redacción, de código fuente, etc.

Por lo expuesto, se agradecerá que en caso de detectar cualquier error lo informe por mail a: ejercicios.aye@gmail.com, detallándolo e indicando el número de ejercicio en donde fue encontrado y el número de versión de esta guía que se indica a continuación:

Versión:
AYED - Modulo 1.5 Cadenas (guia de ejercicios) v2014_2.0.

Los contenidos teóricos necesarios para poder resolver los ejercicios que se proponen en esta guía se encuentran publicados en la siguiente dirección: http://goo.gl/DOSmce., también accesible a través del código QR ubicado a la derecha.

http://goo.gl/UZEXHz

Breve reseña teórica sobre el manejo de cadenas de caracteres

Definiciones

Llamamos "carácter" a un símbolo de nuestro alfabeto y "cadena" a una sucesión finita y ordenada de caracteres.

El carácter

Los caracteres se representan mediante un valor entero positivo de hasta 1 byte (con signo) de longitud. Dicho valor es el que cada carácter tiene asignado en la tabla ASCII. Por ejemplo, al carácter 'A' le corresponde el 65, al carácter 'B' le corresponde el 66 y así sucesivamente. Dicha tabla le asigna el valor 48 al carácter '0', el valor 49 al carácter '1', etcétera.

En C y C++ los caracteres se representan con el tipo de datos `char` que, según lo expuesto más arriba, permite representar en 1 byte de memoria con bit de signo un valor numérico entero.

Luego, las siguientes líneas de código son equivalentes:

```
char c1 = 'A';
char c2 = 65;
```

Y también es válido hacer:

```
int i = 'A';
```

Claro que si decidimos mostrar por consola los valores de las variables `c1`, `c2` e `i` como vemos a continuación:

```
cout << c1 << endl;
cout << c2 << endl;
cout << i << endl;
```

el resultado será el siguiente:

```
A
A
65
```

El hecho de que los caracteres sean tratados como números enteros nos permite realizar algunas operaciones aritméticas. Por ejemplo:

```
char c = '5';
int valorNumericoDeC = c - '0'; // asignamos a la variable su valor numérico: 5
```

La cadena de caracteres

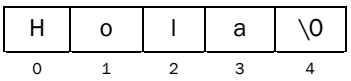
Tanto en C como en C++ no existe el tipo de datos "cadena". En C las cadenas se implementan sobre arrays de caracteres. En C++ existe la clase `string` que, aunque no es un tipo de datos primitivo, nos ofrece una funcionalidad comparable a la que provee el string de Pascal.

Las clases permiten encapsular la lógica de un algoritmo y la complejidad de la estructura de datos que soporta dicha lógica.

Cuando en C++ asignamos:

```
string s = "Hola";
```

internamente se genera una estructura como la siguiente:



Como podemos ver, cada uno de caracteres de la cadena `s` están alojados en celdas numeradas a partir de cero; y al final se agrega un "carácter nulo" que permite distinguir el final de la cadena.

Luego podemos acceder a cada uno de estos caracteres refiriéndonos a `s[i]` donde, en este caso, `i` puede tomar valores entre 0 y la 3. Si accedemos a `s[4]` encontraremos el carácter nulo que se representa como '\0' (barra cero).

La clase `string` permite concatenar cadenas. Veamos el siguiente ejemplo:

```
string s1 = "Hola";
string s2 = ", que tal?";
string s3 = s1 + s2;
cout << s3 << endl; // el resultado sera: Hola, que tal?
```

Biblioteca de funciones

Dado que los ejercicios de esta guía, en su mayoría, consisten en desarrollar diferentes tipos de funciones, será importante conocer cómo podemos agrupar todas estas funciones dentro de un único archivo de código fuente y luego, desde un programa principal, poderlas invocar.

El código QR de la derecha provee acceso a una clase dónde se explica cómo hacerlo.

http://goo.gl/vi5cAA

Guía de ejercicios

Ejercicio 1

Desarrollar y probar adecuadamente la siguiente función:

```
int length(string s);
```

Esta función recibe una cadena y retorna su longitud.

Ejercicio 2

Desarrollar y probar adecuadamente la siguiente función:

```
bool isEmpty(string s);
```

Esta función recibe una cadena y retorna `true` o `false` según se trate o no de la cadena vacía.

http://goo.gl/P4dc4u

Ejercicio 3

Desarrollar y probar adecuadamente la siguiente función:

```
int indexOf(string s, char c);
```

Retorna la posición de la primer ocurrencia del carácter `c` en la cadena `s`; o -1 si `s` no contiene a `c`.

Ejercicio 4

Desarrollar y probar adecuadamente la siguiente función:

```
int lastIndexOf(string s, char c);
```

Retorna la posición de la última ocurrencia del carácter `c` en la cadena `s`; o -1 si `s` no contiene a `c`.

http://goo.gl/dNnwYz

Ejercicio 4.1

Desarrollar y probar adecuadamente la siguiente función:

```
int indexOfN(string s, char c, int n);
```

Retorna la posición de la *n*-ésima ocurrencia del carácter `c` en la cadena `s`; o -1 si `s` no contiene a `c` al menos `n` veces. Dado que `n` hace referencia a la cantidad de ocurrencias, debe comenzar en 1.

Por ejemplo, considerando la siguiente cadena:

```
string s = "Hoy es es mi dia de suerte y todo va a salir a la perfeccion";
```

Entonces:

`indexOfN(s,'e',4)` debe retornar: 22 porque esa es la posición que ocupa la cuarta ocurrencia del carácter 'e' dentro de la cadena `s`.

`indexOfN(s,'e',1)` debe retornar: 4 porque esa es la posición que ocupa la primer ocurrencia del carácter 'e' dentro de la cadena `s`.

`indexOfN(s,'e',8)` debe retornar: -1 porque dentro de `s` el carácter 'e' no aparece 8 veces..

Ejercicio 5

Desarrollar y probar adecuadamente la siguiente función:

```
bool contains(string s, char c);
```

Retorna `true` o `false` según `s` contenga o no al carácter `c`.

http://goo.gl/U46KGT

Ejercicio 6

Desarrollar y probar adecuadamente la siguiente función:

```
int replace(string& s, char oldChar, char newChar);
```

Reemplaza en `s` todas las ocurrencias del carácter `oldChar` por el carácter `newChar` y retorna la cantidad de caracteres reemplazados.

http://goo.gl/1kNNGI

Ejercicio 7

Desarrollar y probar adecuadamente la siguiente función:

```
string substring(string s, int i, int j);
```

Retorna la subcadena de `s` comprendida entre las posiciones `i` (inclusive) y `j` (no inclusive).

Ejercicio 8

Desarrollar y probar adecuadamente la siguiente función:

```
string insertAt(string s, char c, int i);
```

Retorna una cadena de longitud `length(s)+1` compuesta por `s` con el carácter `c` insertado en la *i-ésima* posición.

Ejercicio 9

Desarrollar y probar adecuadamente la siguiente función:

```
string removeAt(string s, int i);
```

Retorna una cadena de longitud `length(s)-1` compuesta por `s` sin el *i-ésimo* carácter.

http://goo.gl/mVk4uw

Ejercicio 10

Desarrollar y probar adecuadamente la siguiente función:

```
string ltrim(string s);
```

Retorna una copia de la cadena `s` pero recortando los espacios en blanco que pudiera tener a la izquierda.

Ejercicio 11

Desarrollar y probar adecuadamente la siguiente función:

```
string rtrim(string s);
```

Retorna una copia de la cadena `s` pero recortando los espacios en blanco que pudiera tener a la derecha.

Ejercicio 12

Desarrollar y probar adecuadamente la siguiente función:

http://goo.gl/CZgiFz

```
string trim(string s);
```

Retorna una copia de la cadena `s` pero sin los espacios en blanco que pudiera tener en los extremos.

Ejercicio 13

Desarrollar y probar adecuadamente la siguiente función:

```
string replicate(int n, char c);
```

Retorna una cadena compuesta de `n` caracteres `c`.

Ejercicio 14

Desarrollar y probar adecuadamente la siguiente función:

```
string spaces(int n);
```

Retorna una cadena compuesta de `n` espacios en blanco.

http://goo.gl/SblYk2

Ejercicio 15

Desarrollar y probar adecuadamente la siguiente función:

```
string lpad(string s,char c, int n);
```

Retorna una cadena de longitud `n` conformada de `n-length(s)` caracteres `c` seguidos de la cadena `s`.

http://goo.gl/INR98I

Ejercicio 16

Desarrollar y probar adecuadamente la siguiente función:

```
string rpad(string s,char c, int n);
```

Retorna una cadena de longitud `n` conformada por la cadena `s` seguida de `n-length(s)` caracteres `c`.

Ejercicio 17

Desarrollar y probar adecuadamente la siguiente función:

```
string cpad(string s,char c, int n);
```

Retorna una cadena de longitud `n` conformada por la cadena `s` y `n-length(s)` caracteres `c` distribuidos uniformemente a la izquierda y a la derecha de ésta.

Ejercicio 18

Desarrollar y probar adecuadamente la siguiente función:

```
string intToString(int n, int v);
```

Retorna una cadena que consiste en la representación del número entero `v` con tantos caracteres '0' a la izquierda como sea necesario agregar para que la longitud total de la cadena sea `n`.

http://goo.gl/iTEWsw

Ejercicio 19

Desarrollar y probar adecuadamente la siguiente función:

```
bool isLetter(char c);
```

Retorna `true` o `false` según el carácter `c` represente o no a una letra.

http://goo.gl/G3nRcf

Ejercicio 20

Desarrollar y probar adecuadamente la siguiente función:

```
bool isUpperCase(char c);
```

Retorna `true` o `false` según el carácter `c` represente o no a una letra mayúscula.

Ejercicio 21

Desarrollar y probar adecuadamente la siguiente función:

```
bool isLowerCase(char c);
```

Retorna `true` o `false` según el carácter `c` represente o no a una letra minúscula.

Ejercicio 22

Desarrollar y probar adecuadamente la siguiente función:

```
char toUpperCase(char c);
```

Retorna en mayúscula el carácter representado por `c`.

Ejercicio 23

Desarrollar y probar adecuadamente la siguiente función:

```
char toLowerCase(char c);
```

Retorna en minúscula el carácter representado por `c`.

Ejercicio 24

Desarrollar y probar adecuadamente la siguiente función:

```
bool isDigit(char c);
```

Retorna `true` o `false` según el carácter `c` represente o no a un dígito numérico.

Ejercicio 25

Desarrollar y probar adecuadamente la siguiente función:

```
int stringToInt(string s);
```

Retorna el valor numérico representado en la cadena `s`. Ejemplo: `stringToInt("2312")` es: 2312.

http://goo.gl/OT4EPb

Ejercicio 26

Desarrollar y probar adecuadamente la siguiente función:

```
int stringToIntN(string s, int n);
```

Retorna el valor numérico representado en la cadena `s` según la base `n`.

A continuación veremos algunos ejemplos.

```
stringToIntN("2312",10);  // retorna: 2312

stringToIntN("4A3F",16);  // retorna: 19007

stringToIntN("4713",8);   // retorna: 2507

stringToIntN("101001",2); // retorna: 41
```

Ejercicio 27

Desarrollar y probar adecuadamente la siguiente función:

```
string doubleToString(int n, double v, int dec);
```

Retorna una cadena de longitud `n` que consiste en la representación del número `v`.

El siguiente ejemplo nos ayudará a comprender el el comportamiento de esta función.

```
doubleToString(10, 25.3, 2); // retorna "0000025.30"
```

Como vemos, la longitud total de la cadena que retorna la función al invocarla con los argumentos del ejemplo es: 10. La parte entera (25) se completó con 5 “ceros a la izquierda” y la parte decimal (3) se completó con 1 “cero a la derecha”.

http://goo.gl/biaA7s

Ejercicio 28

Desarrollar y probar adecuadamente la siguiente función:

```
bool tokenizer(string s,char sep,string& token,int& aux);
```

Sea `s` una cadena y `c` un carácter, entonces llamaremos token a toda subcadena de `s` que se encuentre encerrada entre dos caracteres `c` o entre el inicio de `s` y la primer ocurrencia de `c` o entre la última ocurrencia de `c` y el final de la cadena `s`.

Por ejemplo, si `c` es el carácter '|' y `s` es la cadena:

"John|Paul|George|Ringo"

,los *tokens* que podremos extraer serán: "John", "Paul", "George" y "Ringo", en ese orden. Pero si `c` fuera el carácter 'o', de la misma cadena `s` podríamos extraer los *tokens*; "J", "hn|Paul|Ge", "rge|Ring" y ""; en este caso, el último *token* consiste en una cadena vacía.

http://goo.gl/2HOsck

La función retornará `true` mientras queden *tokens* por extraer y en cada invocación deberá asignar al parámetro `token` el *token* que corresponda.

Veamos un ejemplo.

```
string s = "Juan|Pedro|Pablo|Carlos";
int aux=0;
string token;
while( tokenizer(s,'|',token,aux) )
{
   cout << token << endl;
}
```

La salida será:

```
Juan
Pedro
Pablo
Carlos
```

Ejercicio 29

Desarrollar y probar adecuadamente la siguiente función:

```
int tokenCount(string s, char sep);
```

Retorna la cantidad de *tokens* que el separador `sep` genera en la cadena `s`.

Ejercicio 30

Desarrollar y probar adecuadamente la siguiente función:

```
string getTokenAt(string s, char sep, int i);
```

Dada la cadena `s` y el separador `sep`, retorna el *token* que se ubica en la *i-ésima* posición.

http://goo.gl/0cx3EL

Ejerecicio 30.1

Desarrollar y probar adecuadamente la siguiente función:

```
int findToken(string s, string t, char sep);
```

Dada la cadena `s` *tokenizada* por el separador `sep`, retorna la posición que el *token* `t` ocupa dentro de la cadena o -1 si la misma no contiene a dicho *token*.

Por ejemplo:

```
string beatles = "John|Paul|George|Ringo";
findToken(beatles,"John",'|') debe retornar 0.
findToken(beatles,"Paul",'|') debe retornar 1.
findToken(beatles,"George",'|') debe retornar 2.
findToken(beatles,"Ringo",'|') debe retornar 3.
findToken(beatles,"Mick Jagger",'|') debe retornar -1.
```

Ejerecicio 30.2

Desarrollar y probar adecuadamente la siguiente función:

```
void addToken(string& s, string t, char sep);
```

Agrega el *token* `t` al final de la cadena `s`.

Por ejemplo:

```
string beatles = "";
```

luego de hacer: `addToken(beatles,"John",'|')` la cadena `s` quedará así: `"John"`.

y luego de: `addToken(beatles,"Paul",'|')` la cadena `s` quedará así: `"John|Paul"`.

y luego de: `addToken(beatles,"George",'|')` la cadena `s` quedará así: `"John|Paul|George"`.

y luego de: `addToken(beatles,"George",'|')` la cadena `s` quedará así: `"John|Paul|George|Ringo"`.

Ejerecicio 30.3

Desarrollar y probar adecuadamente la siguiente función:

```
void setTokenAt(string& s, string t, int i, char sep);
```

Asigna el *token* `t` en la posición `i` de la cadena `s` *tokenizada* por el carácter `sep`.

Por ejemplo:

```
string beatles = "John|Paul|George|Ringo";
```

Luego de hacer:

```
setTokenAt(beatles,"Harrison",2,'|');
```

La cadena `beatles` quedará así: “John|Paul|Harrison|Ringo”.

Ejercicio 31

Desarrollar y probar adecuadamente la siguiente función:

```
string reversa(string s);
```

Esta función retorna la cadena inversa de s. Por ejemplo:

```
reversa("arroz",16);  // retorna: "zorra"
```

http://goo.gl/QyT7Ad

Ejercicio 32

Desarrollar y probar adecuadamente la siguiente función:

```
bool capicua(string s);
```

Esta función retorna `true` o `false` según s sea o no una cadena capicúa.

Ejercicio 33

Se desea saber qué cantidad de billetes de cada denominación se deben utilizar para pagar un importe determinado. El importe y el conjunto de denominaciones disponibles se ingresarán por consola de la siguiente manera:

http://goo.gl/TnW959

Ingrese importe: 2342 *[ENTER]*

Ingrese denominaciones: 100,50,20,10,5,2,1 *[ENTER]*

Según este ejemplo, el programa debe arrojar el siguiente resultado:

- 23 billetes de $100
- 2 billetes de $20
- 1 billete de $2

En cambio, si se ingresan los siguientes datos:

Ingrese importe: **2342** *[ENTER]*

Ingrese denominaciones: **80,40,30,5,1** *[ENTER]*

El programa debería mostrar lo siguiente:

- 29 billetes de $80
- 1 billetes de $40
- 2 billetes de $1

Ejercicio 34

Desarrolle un programa que permita optimizar la cantidad de billetes con los que se debe abonar una determinada suma de dinero, priorizando el hecho de que la cantidad de billetes a entregar sea mínima; las denominaciones a utilizar serán: $50, $40, $20, $10, $5, $2 y $1.

Por ejemplo: si la suma a abonar fuese: $80 el programa debería sugerir abonar ese importe utilizando 2 billetes de $40 en lugar de utilizar 1 billete de $50, 1 billete de $20 y 1 de $10.

http://goo.gl/hbilNl