ALGORITMOS Y ESTRUCTURA DE DATOS

Guía Práctica
Cadenas de caracteres

Breve reseña teórica sobre 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

```
2
```

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:

H	o	l	a	\0
0	1	2	3	4

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, llamado “barra cero”, 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?
```

Funciones básicas

Función: **length**.

Objetivo: Retornar la longitud de la cadena `s` que recibe como parámetro.

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

Función: **charCount**.

Objetivo: Retornar la cantidad de veces que la cadena `s` contiene al carácter `c`.

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

Función: **substring**.

Objetivo: Retornar la subcadena de `s` comprendida entre las posiciones `d` (inclusive) y `h` (no inclusive).

```
string substring(string s, int d, int h);
```

Función: **substring** (sobrecarga).

Objetivo: Retornar la subcadena de `s` comprendida entre la posición `d` (inclusive) y el final de la cadena.

Nota: Resolver invocando a la función `length` y a la versión anterior de `substring`.

```
string substring(string s, int d);
```

Función: **indexOf**.

Objetivo: Retornar la posición de la primera ocurrencia de `c` dentro de `s`, o -1 si `s` no contiene a `c`.

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

Función: **indexOf** (sobrecarga).

Objetivo: Idéntico a la anterior, pero descartando la subcadena de `s` comprendida 0 y `offset` (no inclusive).

Nota: Resolver invocando a `substring` y a la versión anterior de `indexOf`.

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

Función: **indexOfN**.

Objetivo: Retornar la posición de la *enésima* ocurrencia de `c` dentro de `s`.

Nota: Resolver invocando a `indexOf`.

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

Función: **pow**.

Objetivo: Calcular y retornar la potencia que resulta de elevar `x` a la `y`.

```
int pow(int x, int y);
```

```
4
```

Función: **charToInt**.

Objetivo: Retornar el valor numérico representado por el carácter `c`.

```
int charToInt(char c);
```

Función: **intToChar**.

Objetivo: Retornar el carácter que representa al valor numérico `i`.

```
char intToChar(int i);
```

Función: **getDigit**.

Objetivo: Retornar el *i*-ésimo dígito de n, comenzando desde la derecha.

```
int getDigit(int n, int i);
```

Función: **digitCount**.

Objetivo: Retornar la cantidad de dígitos que tiene el valor entero `i`.

```
int digitCount(int i);
```

Función: **intToString**.

Objetivo: Retornar una cadena de caracteres que represente al valor del entero `i`.

Nota: Resolver invocando a `digitCount`, `getDigit` e `intToChar`.

```
string intToString(int i);
```

Función: **stringToInt**.

Objetivo: Retornar el valor entero que, en base numérica `b`, está representado en la cadena `s`.

Nota: Resolver invocando a `length`, `charToInt` y `pow`.

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

Función: **stringToInt** (sobrecarga).

Objetivo: Retornar el valor entero que, en base numérica `10`, está representado en la cadena `s`.

Nota: Resolver invocando a la versión anterior de `stringToInt`.

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

Función: **charToString**.

Objetivo: Retornar una cadena de longitud 1 cuyo único carácter sea `c`.

```
string charToString(char c);
```

Función: **stringToChar**.

Objetivo: Retornar el primer carácter de la cadena `s`.

```
char stringToChar(string s);
```

Función: **doubleToString**.

Objetivo: Retornar una cadena de caracteres que represente al valor numérico `d`.

```
string doubleToString(double d);
```

Función: **stringToDouble**.

Objetivo: Retornar el valor numérico representado en la cadena `s`.

```
double stringToDouble(string s);
```

Funciones adicionales

Función: **isEmpty**.

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

Nota: Resolver en una sola línea invocando a `length`.

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

Función: **contains**.

Objetivo: Retorna true o `false` según se `s` contenga o no a `c`.

Nota: Resolver en una sola línea invocando a `indexOf`.

```
6
```

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

Función: **replace**.

Objetivo: Retorna una cadena idéntica a `s` pero reemplazando todas las ocurrencias de `oldChar` por `newChar`.

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

Función: **insertAt**.

Objetivo: Retorna una cadena idéntica a s pero insertando el carácter `c` en la posición `pos` de la nueva cadena.

Nota: Resolver usando `substring`.

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

Función: **removeAt**.

Objetivo: Retorna una cadena idéntica a s pero eliminando el de la posición `pos`.

Nota: Resolver usando `substring`.

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

Función: **ltrim**.

Objetivo: Retorna una cadena idéntica a `s` pero quitando todos los espacios en blanco de la izquierda.

Nota: Resolver usando `substring`.

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

Función: **rtrim**.

Objetivo: Retorna una cadena idéntica a `s` pero quitando todos los espacios en blanco de la derecha.

Nota: Resolver usando `substring`.

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

Función: **trim**.

Objetivo: Retorna una cadena idéntica a `s` pero sin espacios en blanco en los extremos.

Nota: Resolver en una sola línea.

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

Función: **replicate**.

Objetivo: Retorna una cadena compuesta de `n` en caracteres `c`.

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

Función: **spaces**.

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

Nota: Resolver invocando a `replicate`.

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

Función: **lpad**.

Objetivo: Retorna una cadena idéntica a `s` pero de longitud `n` complentando, de ser necesario, con caracteres `c` a la izquierda.

Nota: Resolver invocando a `length` y `replicate`.

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

Función: **rpad**.

Objetivo: Retorna una cadena idéntica a `s` pero de longitud `n` complentando, de ser necesario, con caracteres `c` a la derecha.

Nota: Resolver invocando a `length` y `replicate`.

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

Función: **cpad**.

Objetivo: Retorna una cadena idéntica a `s` pero de longitud `n` complentando, de ser necesario, con caracteres `c` distribuidos equitativamente a izquierda y a derecha.

```
 8
```

Nota: Resolver invocando a `lpad` y `rpad`.

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

Función: **isDigit**.

Objetivo: Retorna `true` o `false` según `c` sea o no un dígito (de 0 a 9).

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

Función: **isLetter**.

Objetivo: Retorna `true` o `false` según `c` sea o no una letra.

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

Función: **isUpperCase**.

Objetivo: Retorna `true` o `false` según `c` sea o no una letra en mayúscula.

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

Función: **isLowerCase**.

Objetivo: Retorna `true` o `false` según `c` sea o no una letra en munúscula.

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

Función: **toUpperCase**.

Objetivo: Retorna el carácter `c` pero en mayúscula.

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

Función: **toLowerCase**.

Objetivo: Retorna el carácter `c` pero en munúscula.

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

Tratamiento de tokens

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` fuese el carácter 'o', considerando la misma cadena `s`, los tokens a extraer serán: "J", "hn|Paul|Ge", "rge|Ring" y ""; el último *token* consiste en la cadena vacía

Función: **tokenCount**.

Objetivo: Retornar la cantidad de *tokens* que surgen de la cadena `s` considerando al separador `sep`.

Nota: Resolver invocando a `charCount`.

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

Función: **getTokenAt**.

Objetivo: Retornar el *i*-ésimo *token* que surge de s al separarla por el carácter `sep`.

Nota: Resolver invocando a `indexOfN`, `tokenCount` y `substring`.

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

Función: **addToken**.

Objetivo: Agregar el token `t` al final de la cadena `s`.

Nota: Resolver invocando a `tokenCount`.

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

Función: **removeTokenAt**.

Objetivo: Remover de la cadena `s` el *i*-ésimo *token*.

Nota: Resolver invocando a `tokenCount`, `getTokenAt` y `addToken`.

```
string removeTokenAt(string& s, char sep, int i);
```

```
10
```

Función: **setTokenAt**.

Objetivo: Reemplazar con `t` el *i*-ésimo *token* de la cadena `s`.

Nota: Resolver invocando a `tokenCount`, `getTokenAt` y `addToken`.

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

Función: **findToken**.

Objetivo: Determinar si la cadena `s` contiene al *token* `t`, retornando la posición de su primer ocurrencia o -1 en caso de que `s` no contenga a `t`.

Nota: Resolver invocando a `tokenCount` y `getTokenAt`.

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

Colecciones de datos | TAD Coll

Estructura

```
struct Coll
{
   string s;
   char sep;
}
```

Descripción

Este TAD representa una colección de elementos de cualquier tipo de dato: `string`, `int`, `double` y `char`; incluso, tipos definidos por el programador.

La implementación debe realizarse mediante el tratamiento de *tokens*.

Funciones Generales

- `Coll collCreate(char sep)` | Retorna una colección vacía.
- `int collSize(Coll c)` | Retorna la cantidad de elementos de la colección.
- `void collRemoveAll(Coll& c)` | Remueve todos los elementos de la colección dejándola vacía.
- `void collRemoveAt(Coll& c, int p)` | Remueve el elemento ubicado en la posición `p`.

Funciones para valores de cadena (`string`)

- `void collAddString(Coll& c, string s)` | Agrega la cadena `s` al final de la colección `c`.
- `void collSetStringAt(Coll& c, string s, int p)` | Reemplaza con `s` al valor de la posición p.
- `string collGetStringAt(Coll c, int p)` | Retorna el valor ubicado en la posición `p`.
- `int collFindString(Coll c, string s)` | Retorna la posición de la primer ocurrencia de `s` o -1.

Funciones para valores enteros (`int`)

- `void collAddInt(Coll& c, int i)` | Agrega el valor entero `i` al final de la colección.
- `void collSetIntAt(Coll& c, int i, int p)` | Reemplaza con `i` al valor ubicado en la posición `p`.

- `int collGetIntAt(Coll c, int p)` | Retorna el valor entero ubicado en la posición `p`.
- `int collFindInt(Coll c, int i)` | Retorna la posición de la primer ocurrencia de `i` o -1.

Funciones para valores flotantes (`double`)

- `void collAddDouble(Coll& c, double d)`
- `void collSetDoubleAt(Coll& c, double d, int p)`
- `double collGetDoubleAt(Coll c, int p)`
- `int collFindDouble (Coll c, double d)`

Funciones para valores carácter (`char`)

- `void collAddChar(Coll& c, char ch)`
- `void collSetCharAt(Coll& c, char ch, int p)`
- `char collGetCharAt(Coll c, int p)`
- `int collFindChar(Coll c, char ch)`

Funciones para valores genéricos (`T`)

- ```
  template <typename T>
  void collAdd<T>(Coll& c, T t, string tToString(T))
  ```

  Agrega un valor de tipo `T` al final de la colección. La función `tToString` que se recibe como parámetro recibe, a su vez, un parámetro de tipo `T` y debe retornar una cadena de caracteres que permita representarlo.

- ```
  template <typename T>
  void collSetAt<T>(Coll& c, T t, int p, string tToString(T))
  ```

  Reemplaza por `t` al elemento que se ubica en la posición `p` de la colección. La función `tToString` debe retornar una cadena que represente al valor de tipo `T` que recibe como parámetro.

- ```
  template <typename T>
  T collGetAt(Coll c, int p, T stringToT(string))
  ```

  Retorna el elemento de tipo `T` que se ubica en la posición `p` de la colección. La función `stringToT`, que se recibe como parámetro, debe retornar un valor de tipo `T` a partir de una cadena.

- ```
  template <typename T, typename K>
  int collFind(Coll c, K k, int compare(T,K), T stringToT(string) )
  ```

  Retorna la posición de la primer ocurrencia de `k`, de tipo `K`, dentro una colección de valores de tipo `T`. La función `stringToT`, que recibe como parámetro, ya fue explicada más arriba. La función compare debe comparar dos valores de tipo `T` y `K` respectivamente; y retornar un valor entero que será: menor, igual o mayor que 0 (cero) según se considere que `t` sea: menor, igual o mayor que `k`.

- ```
  template <typename T >
  void collSort(Coll& c
                ,T t
                ,int compare(T,T)
                ,T stringToT(string)
                ,string tToString(T));
  ```

Ordena la colección `c` según el criterio de comparación que establece la función compare.

Ejemplos

Colección de enteros

```
int main()
{
   Coll c = collCreate('|');
   collAddInt(c,1);
   collAddInt(c,2);
   collAddInt(c,3);
   for(int i=0; i<collSize(c); i++)
   {
      int v = collGetIntAt(c,i);
      cout << v << endl;
   }
   return 0;
}
```

Colección de cadenas

```
int main()
{
   Coll c = collCreate('|');
   collAddString(c,"uno");
   collAddString(c,"dos");
   collAddString(c,"tres");
   for(int i=0; i<collSize(c); i++)
   {
      string v = collGetStringAt(c,i);
      cout << v << endl;
   }
   return 0;
}
```

Colección de objetos (tipos genéricos)

```
struct Persona
{
   int dni;
   string nombre;
}
Persona personaCreate(int dni, string nom)
{
   Persona p;
   p.dni = dni;
   p.nombre = nom;
   return p;
}
string personaToString(Persona p)
{
   return intToString(p.dni)+","+p.nombre;
}
Persona stringToPersona(string s)
{
   Persona p;
   p.dni = stringToInt(getTokenAt(s,0,','));
   p.nombre = getTokenAt(s,1,','));
   return p;
}
```
```

```
int personaCompareDNI(Persona p, int dni)
{
   return p.dni-dni;
}
int main()
{
   Coll c = collCreate('|');
   collAdd<Persona>(c,personaCreate(10,"Pedro"),personaToString);
   collAdd<Persona>(c,personaCreate(20,"Pablo"),personaToString);
   collAdd<Persona>(c,personaCreate(30,"Juan"),personaToString);
   for(int i=0; i<collSize(c); i++)
   {
      Persona v = collGetAt<Persona>(c,i,stringToPersona);
      cout << v.dni << "," << v.nombre << endl;
   }

   int dni;
   cout << "Ingrese un dni: ";
   cin >> dni;

   // busco una persona x DNI... la encuentro en la posicion: pos
   int pos = collFind<Persona,int>(c,dni,personaCompareDNI,stringToPersona);

   // obtengo la persona ubicada en la posicion pos de la coleccion
   Persona pers = collGetAt<Persona>(c,pos,stringToPersona);

   cout << pers.dni << "," << pers.nombre << endl;
   return 0;
}
```