ESTRUCTURAS DE DATOS

PRÁCTICA 1 – ÁBACO

CURSO 2017 - 18

ESTEFANÍA MARTÍN ROJAS
73133122-E
Universidad de Alcalá de Henares

1. Detalles y justificación de la implementación:

1.1 Especificación concreta de la interfaz de los TAD's implementados.

1.1.1 TAD's creados.

- Lista
- Nodo de la lista
- Pila
- Nodo de la pila
- Bolita
- Abaco

1.1.2 Definición de las operaciones TAD (Nombre, argumentos y retorno).

- **Lista:**
  - Lista();
  - ~Lista();
  - void insertarNodo(Pila columna, char nombre, char tipo);
  - bool comprobarExistenciaColumna(char buscado);
  - bool listaVacia ();
  - void borrarActual();
  - bool borrarColumna(char buscado);
  - void mostrarLista();
  - void LmeterBolas(char buscado, string color);
  - void apilarActual(string color);
  - void LsacarBolas(char buscado);
  - void desapilarActual();
  - int LtamanoPila(char buscado);
  - int tamanoActual();
- **Nodo de la lista:**
  - NodoLista(Pila colum, char nom);
  - char getNombre();
  - Pila getColumna();
- **Pila**:
  - Pila();
  - ~Pila();
  - void apilar(string color);
  - void desapilar();
  - bool esVacia();
  - string verCima();
  - int tamanoPila();
  - pnodoPila getcima();
- **Nodo de la pila:**
  - NodoPila(string colorBola, NodoPila *sig);
  - string getColorPila();
- **Bolita:**
  - Bolita();
  - Bolita(string color);
  - string getColor();

- **Abaco:**
    - abaco();
    - void introducirColumnas();
    - void introducirBolitasEnAbaco(int un,int de,int ce,int unm, int dem);
    - int separar_numero(int& numero, int& dm, int& um, int& c, int& d, int& u);
    - bool separar_hexadecimal(string hexa, int& dm, int& um, int& c, int& d, int& u);
    - void verEstadoPilas();
    - void suma(int u2, int d2, int c2, int mu2, int md2, int bolas);
    - void resta(int u2, int d2, int c2, int mu2, int md2, int bolas);
    - void vaciarLista();
    - void sumaPila(int bolasMeter, char pila, char pilaSiguiente, int bolas);
    - void restaPila(int bolasSacar, char pila, char pilaSiguiente, int bolas);

1.2 Solución adoptada: explicación de la implementación.

Para poder realizar esta práctica, he creado un ábaco que realmente es una lista. Los nodos del ábaco, es decir, de la lista, son pilas que corresponden con cada columna del ábaco. Estas columnas contienen ocho, diez o dieciséis bolas, dependiendo de lo que elija el usuario en el menú del programa que se aloja en la clase main().

Desde la lista, se llaman a operaciones de las pilas que ésta contiene para realizar una suma o una resta gracias a los punteros que he creado.

1.3 Explicación de los métodos más destacados.

En cuanto a la clase **Lista**, los más importantes son los siguientes:

- insertarNodo: gracias a este método, podemos crear una columna en el ábaco que después contendrá bolitas que podremos meter o sacar.
- LmeterBolas/LsacarBolas: gracias a este método, desde la lista, podemos meter o sacar bolas en una pila que nosotros queramos.
- LtamanoPila: gracias a este método sabremos cuantas bolas hay en una pila.

En cuanto a la **Pila**, los más importantes son:

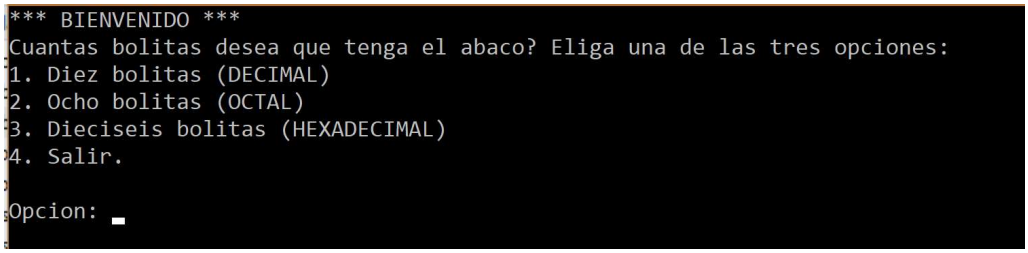
- Apilar/desapilar: puede meter o sacar bolitas.
- tamanoPila: devuelve el número de bolitas que hay en la pila.

Por último, en la clase **MAS IMPORTANTE** que es **Abaco**, los métodos más importantes son:

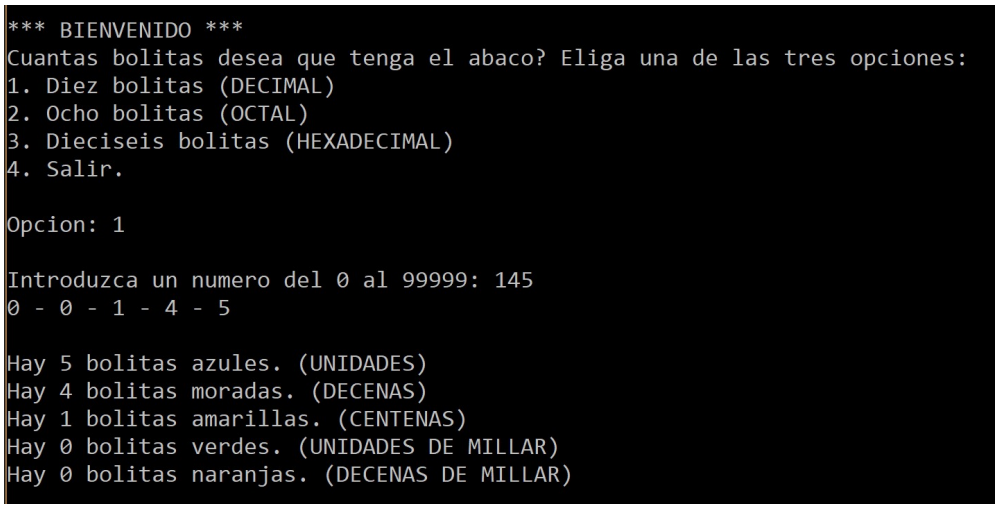
- separar_numero/separar_hexadecimal: en este método introducimos un numero entero y nos devuelve ese número partido en unidades, decenas, centenas… Esencial para saber cuántas bolitas corresponden a cada columna.
- Suma/resta: son las operaciones que puede realizar el ábaco.
- VaciarLista(): al final de cada operación realizada por el ábaco (suma o resta), se destruye el ábaco.

1.4 Manual de usuario.

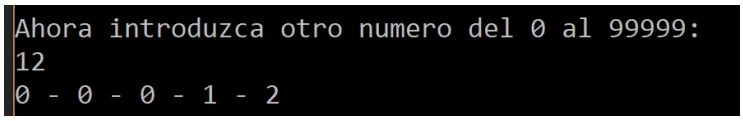
Al iniciar el programa, se presenta un menú donde se pide cuántas bolitas desea el usuario que tenga el ábaco en cada columna. Podemos elegir diez, ocho o dieciséis bolas:



Una vez que el usuario elija el número de bolas, se le pedirá un primer número. Recuerde que el primer número debe ser MAYOR que el segundo número que se pedirá después. Este primer número se separará en cinco partes, cada una corresponderá con una columna de la pila y cada valor de cada parte será el número de bolitas que habrá en cada columna:



A continuación, se le pedirá al usuario un segundo número y hará el mismo progreso que con el primer número, sin embargo, el valor de este número no se meterá en el ábaco. Servirá para la suma o la resta.



Por último, el usuario deberá elegir si quiere sumar o restar estos dos números. El programa mostrará las bolitas que han quedado en el ábaco, es decir, el resultado de la cuenta:

```
Que desea hacer?:
1. Suma
2. Resta

Opcion: 1
Hay 7 bolas.
Hay 5 bolas.
Hay 1 bolas.
Hay 0 bolas.
Hay 0 bolas.
Presione una tecla para continuar . . .
```

Por último, el programa pedirá presionar cualquier tecla, para volver a realizar otra consulta.

2. Ejemplos de funcionamiento. Pruebas.

Se han realizado múltiples pruebas. Algunas pruebas el programa no las pasa.

Por ejemplo, si realiza la siguiente resta: 100-1, no la realiza correctamente, ya que hay cero bolitas y al quitar una decena porque en las unidades nos llevamos una, no deja nueve bolas, sino cero. Es un error que pude solucionar, sin embargo, la centena se quedaba intacta ya que no supe solucionar que el uno que queda del cien (la centena), fuera uno menos.

La solución más rápida sería poner un tope de nodos a la pila. De diez, ocho o dieciséis, sin embargo, no he dispuesto de tiempo para ello y hacer pruebas al respecto.

```
Opcion: 1

Introduzca un numero del 0 al 99999: 100
0 - 0 - 1 - 0 - 0

Hay 0 bolitas azules. (UNIDADES)
Hay 0 bolitas moradas. (DECENAS)
Hay 1 bolitas amarillas. (CENTENAS)
Hay 0 bolitas verdes. (UNIDADES DE MILLAR)
Hay 0 bolitas naranjas. (DECENAS DE MILLAR)

Ahora introduzca otro numero del 0 al 99999:
1
0 - 0 - 0 - 0 - 1
Que desea hacer?:
1. Suma
2. Resta

Opcion: 2
Hay 9 bolas.
Hay 0 bolas.
Hay 1 bolas.
Hay 0 bolas.
Hay 0 bolas.
Presione una tecla para continuar . . .
```

Por otro lado, también hice pruebas con la opción hexadecimal. El programa pide un STRING en vez de un entero como hago en las otras dos opciones. Al querer separar el string, comparo cada char y lo igualo a su correspondiente número en hexadecimal. El problema es que, si el número no es de 5 dígitos, confunde los valores de las unidades, decenas y demás, ya que tengo configurado que las unidades sean el quinto lugar del string. Si introducimos un numero de 4 dígitos, las unidades serás las decenas, las decenas las centenas y así, dando errores en la suma y en la resta.

- Correcto:

```
Opcion: 3
Introduzca el primer numero:
14AB3
1 - 11 - 3

Hay 3 bolitas azules. (UNIDADES)
Hay 11 bolitas moradas. (DECENAS)
Hay 10 bolitas amarillas. (CENTENAS)
Hay 4 bolitas verdes. (UNIDADES DE MILLAR)
Hay 1 bolitas naranjas. (DECENAS DE MILLAR)
```

- Error:

```
Opcion: 3
Introduzca el primer numero:
23A
2 - 11 - 3

Hay 3 bolitas azules. (UNIDADES)
Hay 11 bolitas moradas. (DECENAS)
Hay 10 bolitas amarillas. (CENTENAS)
Hay 3 bolitas verdes. (UNIDADES DE MILLAR)
Hay 2 bolitas naranjas. (DECENAS DE MILLAR)
```

En el menú si el usuario introduce un número que no corresponda con alguna opción existente, se reinicia.

En la opción de los octales, si algún número introducido contiene un 8 o 9, da error y vuelve a pedir el número.

Si se introduce el primer número mayor que el segundo y se realiza la resta, el programa se reinicia.

3. Bibliografía.

- Separar un numero en unidades, decenas…: https://youtu.be/T0kRKkQFk7s
- Restas con llevadas: https://www.youtube.com/watch?v=WQu3gMc7D1s
- Dudas sobre el lenguaje c++: http://www.cplusplus.com/reference/
- Dudas sobre listas y pilas: documentación de la universidad y http://c.conclase.net/edd/?cap=000#inicio