Ejercicios de Pilas

Información del Proyecto

Descripción	Detalles
Profesores	Sergio Cavero y Salvador Sanchez
Asignatura	Estructuras de Datos
Universidad	Universidad Rey Juan Carlos
Licencia	CC BY-NC-SA 4.0

¿Cómo enfrentarse a los ejercicios?

Antes de comenzar con cada uno de los ejercicios:

1. Lee detenidamente el enunciado **completo**.
2. Identifica cuál es el objetivo del ejercicio:
    - Identificar el tipo de elementos que se amlacenará en la pila:
        - Ejemplos: enteros, caracteres, registros, etc.
        - Esto va a determinar la definición de los nodos de la pila (o el tipo de array en caso de no ser una pila dinámica).
        - Es decir, deberemos **trabajar en la propia unidad** de la pila. Tendremos el ``rol'' de implementador de la unidad.
    - Extender o modificar funcionalidades de la pila:
        - Ejemplos: contar el número de elementos de la pila, verificar si un elemento está en la pila, eliminar un elemento, etc.
        - Generar nuevos procedimientos o funciones requerirá utilizar las operaciones principales.
        - Es decir, nuevamente deberemos **trabajar en la propia unidad** de la pila ya que estamos extendiendo su funcionalidad. Tendremos el ``rol'' de implementador de la unidad.
    - Utilizar la pila para resolver un problema específico:
        - Ejemplos: verificar si una expresión aritmética está balanceada, verificar si una palabra es palíndroma, utilizarla para representar un conjunto de elementos, etc.
        - En este caso, deberemos utilizar las operaciones básicas de la unidad de la pila para resolver el problema, no podemos modificar la definición de la pila.
        - Por lo tanto, deberemos **usar** la pila en el programa principal para resolver el problema. Tendremos el rol de usuario externo de la unidad pila y por lo tanto no la podremos modificar.
3. Abre cada uno de los archivos proporcionados y estudia el código proporcionado.
4. Haz que el programa compile y ejecuta el programa aunque no haga lo que se pide.

5. Realiza los ejercicios de manera ordenada, comprobando que cada uno de ellos funciona correctamente antes de pasar al siguiente.

Ejercicio 1: Operaciones balanceadas

En este ejercicio, trabajaremos con operaciones básicas de pilas. Para ello, se proporcionan dos archivos Pascal:

- `uPilaChar.pas`
- `pilas_ej1.pas`

Ejercicio 1.1 Implememtentar un TAD Pila de Caracteres en `uPilaChar.pas`

Este archivo contiene la definición de la pila de caracteres pero no su implementación. La pila de caracteres es un TAD que da soporte al concepto de una pila que almacena caracteres y por tanto, tiene las operaciones básicas de una pila:

- **Inicializa la pila**

```
procedure initialize(var p: tPilaChars);
```

- **Agrega un elemento a la pila**

```
procedure push(var p: tPilaChars; x: char);
```

- **Elimina el elemento de la cima de la pila**

```
procedure pop(var p: tPilaChars);
```

- **Devuelve el elemento de la cima de la pila**

```
function peek(p: tPilaChars): char;
```

- **Devuelve true si la pila está vacía**

```
function isEmpty(p: tPilaChars): boolean;
```

Tu cometido en este punto es implementar todas las funciones básicas de la Pila de Caracteres. Puedes elegir para su implementación el modelo de pila implementada con un array o pila con punteros y memoria

dinámica. Se recomienda este último modelo, y se sugiere que el del array se haga también como trabajo extra en casa.

Ejercicios 1.2 y 1.3 en `pilas_ej1.pas`

En este archivo, los estudiantes deben desarrollar dos funciones:

1.2 balanceada:

Define una función `balanceada` que, utilizando una pila de caracteres, compruebe el balance de paréntesis de una expresión aritmética. Una expresión aritmética está balanceada si cada paréntesis de apertura tiene un paréntesis de cierre correspondiente. Los paréntesis deben estar correctamente anidados. No se comprobará si los paréntesis están en el lugar correcto (por ejemplo, `(3+2*)5` o `3(+4-)2` no es una expresión aritmética válida pero vamos a considerarla balanceada).

- **Entradas**:
  - Una cadena de caracteres que representa una expresión aritmética
- **Salida**:
  - Devuelve `true` si la expresión está balanceada, es decir, si los paréntesis están correctamente cerrados y anidados.
- **Ejemplos**:
  - `(3+2)*5` -> `true`
  - `3+(4-2` -> `false`
  - `(3+2)*5+(4-2)` -> `true`
  - `(3(+2)*5)+(4-2` -> `false`

1.3 balanceada2:

Mejora la función `balanceada`, utilizando una pila de caracteres, para que compruebe también el balance de corchetes. Una expresión aritmética está balanceada si cada paréntesis de apertura tiene un paréntesis de cierre correspondiente y cada corchete de apertura tiene un corchete de cierre correspondiente.

- **Entradas**:
  - Una cadena de caracteres que representa una expresión aritmética.
- **Salida**:
  - Devuelve `true` si la expresión está balanceada, es decir, si los paréntesis y corchetes están correctamente cerrados y anidados.
- **Ejemplos**:
  - `(3+2)*5` -> `true`
  - `3+(4-2]` -> `false`
  - `(3+2)*5+(4-2)` -> `true`
  - `(3(+2)*5)+(4-2` -> `false`
  - `[3+2]*5` -> `true`
  - `3+[4-2` -> `false`
  - `[3+2]*5+[4-2]` -> `true`
  - `[3[+2]*5]+[4-2]` -> `false`

Comprobar el correcto funcionamiento de las funciones implementadas

En el programa principal, `pilas_ej1.pas`, se proporcionan 11 casos de prueba. Una vez hayas implementado las funciones de los ejercicios 1.2 y 1.3 en `pilas_ej1.pas` podrás ejecutar este programa principal y comprobar tu código. Cuando se ejecute, se mostrará una tabla como la siguiente:

```
Ejemplo #  Expresión            Res = Esp    OK/ERROR
-----------------------------------------------------
Ejemplo 1: (3+2)*5              TRUE = TRUE  OK
Ejemplo 2: 3+(4-2               FALSE = FALSE OK
Ejemplo 3: (3+2)*5+(4-2)        TRUE = TRUE  OK
Ejemplo 4: (3(+2)*5)+(4-2       FALSE = FALSE OK
Ejemplo 5: [3+2]*5              TRUE = TRUE  OK
Ejemplo 6: 3+[4-2               FALSE = FALSE OK
Ejemplo 7: [3+2]*5+[4-2]        TRUE = TRUE  OK
Ejemplo 8: [3[+2]*5]+[4-2)      FALSE = FALSE OK
Ejemplo 9: [3[+2]*5)+[4-2]      FALSE = FALSE OK
Ejemplo 10: (3[+2]*5(+[4-2]     FALSE = FALSE OK
Ejemplo 11: (3[+2]*5)+[4-2]     TRUE = TRUE  OK
```

Ejercicio 2: Operaciones avanzadas con pilas dinámicas

En este ejercicio vamos a tomar el rol de implementador de un TAD pila y, a partir de una unit ya implementada, vamos a ampliarla añadiéndole operaciones avanzadas de pilas. Para ello, se proporcionan dos archivos Pascal:

- `uPilaIntegerExtended.pas`
- `pilas_ej2.pas`

El archivo `uPilaIntegerExtended.pas` contiene la definición de la pila de enteros extendida. Por otro lado, el archivo `pilas_ej2.pas` es el programa principal que se utilizará para comprobar el correcto funcionamiento de las funciones implementadas.

Concretamente, las operaciones que queremos implementar con la pila de enteros extendida en el archivo `uPilaIntegerExtended.pas` son:

2.1 contarElementos:

Define una función `contarElementos` para contar el número de elementos en una pila. La operación debe realizarse en O(1). Puede que necesites modificar el registro tPilaEnterosExt para ello.

- **Entradas**:
  - Pila de enteros.
- **Salida**:
  - Devuelve la cantidad de elementos en la pila.
- **Ejemplos**:
  - `[1, 2, 3, 4, 5]` -> `5`
  - `[]` -> `0`

- `[1, 2, 3]` -> `3`

2.2 ultimo:

Define una función `ultimo` para obtener el elemento en el fondo de la pila, sin modificarla. Es decir, el entero que se encuentra sumergido en el extremo opuesto de la cima de la pila, por lo que se podría entender esta función como una operación opuesta a "peek" en la pila.

- **Entradas**:
  - Pila de enteros.
- **Salida**:
  - Devuelve el elemento en la última posición de la pila.
- **Ejemplos**:
  - `[1, 2, 3, 4, 5]` -> `5`
  - `[]` -> `indeterminado`
  - `[1, 2, 3]` -> `3`

2.3 combinar:

Define un procedimiento `combinar` para combinar dos pilas en una sola.

- **Entradas**:
  - Dos pilas de enteros.
- **Salida**:
  - Modifica la primera pila con los elementos de ambas pilas. El orden de los elementos de la segunda pila debe ser el mismo que el original. La segunda pila queda vacía.
- **Ejemplos**:
  - `combinar([1, 2, 3], [4, 5, 6])` -> La primera pila pasa a ser: `[4, 5, 6, 1, 2, 3]`
  - `combinar([], [4, 5, 6])` -> La primera pila pasa a ser: `[4, 5, 6]`
  - `combinar([1, 2, 3], [])` -> La primera pila pasa a ser: `[1, 2, 3]`

2.4 popN:

Define un procedimiento `popN` para hacer "pop" n veces.

- **Entradas**:
  - Pila de enteros, entero n.
- **Salida**:
  - Modifica la pila eliminando (un máximo de) n elementos de la cima. Si la pila contiene menos de n elementos, elimina todos los que haya.
- **Ejemplos**:
  - `popN([1, 2, 3, 4, 5], 2)` -> `[3, 4, 5]`
  - `popN([1, 2, 3, 4, 5], 20)` -> `[]`
  - `popN([1, 2, 3, 4, 5], 0)` -> `[1, 2, 3, 4, 5]`

2.5 sumarN:

Define un procedimiento `sumarN` para sumar los n primeros elementos de una pila y reemplazarlos por su suma.

- **Entradas**:
  - Pila de enteros, entero n.
- **Salida**:
  - Modifica la pila reemplazando los n elementos por la suma de los mismos. Si la pila contiene menos de n elementos, suma todos los que haya, reemplazándolos por su suma.
- **Ejemplos**:
  - `sumarN([1, 2, 3, 4, 5], 2)` -> `[3, 3, 4, 5]`
  - `sumarN([1, 2, 3, 4, 5], 5)` -> `[15]`
  - `sumarN([1, 2, 3, 4, 5], 8)` -> `[15]`
  - `sumarN([1, 2, 3, 4, 5], 0)` -> `[1, 2, 3, 4, 5]`

2.6 invertir:

Define un procedimiento `invertir` para invertir los elementos de una pila.

- **Entradas**:
  - Pila de enteros.
- **Salida**:
  - Modifica la pila invirtiendo el orden de sus elementos.
- **Ejemplos**:
  - `invertir([1, 2, 3, 4, 5])` -> `[5, 4, 3, 2, 1]`
  - `invertir([])` -> `[]`
  - `invertir([1, 2, 3])` -> `[3, 2, 1]`

2.7 repetirN:

Define un procedimiento `repetirN` para repetir n veces los elementos de la pila.

- **Entradas**:
  - Pila de enteros, entero n (n >= 1).
- **Salida**:
  - Modifica la pila repitiendo n veces los elementos.
- **Ejemplos**:
  - `repetirN([1, 2, 3], 2)` -> `[1, 1, 2, 2, 3, 3]`
  - `repetirN([1, 2], 3)` -> `[1, 1, 1, 2, 2, 2]`
  - `repetirN([1, 2, 3], 1)` -> `[1, 2, 3]`

2.8 contarApariciones:

Define una función `contarApariciones` para contar las apariciones de un elemento en la pila.

- **Entradas**:
  - Pila de enteros, entero n.
- **Salida**:
  - Devuelve la cantidad de veces que aparece el elemento n en la pila.
- **Ejemplos**:
  - `contarApariciones([1, 2, 3, 4, 5], 2)` -> `1`
  - `contarApariciones([1, 2, 3, 4, 5], 10)` -> `0`
  - `contarApariciones([1, 2, 3, 4, 5, 2], 2)` -> `2`

Comprobar el correcto funcionamiento de las funciones implementadas

En el programa principal, `pilas_ej2.pas`, se proporcionan funciones que realizan pruebas de las funciones implementadas. Cuando se ejecute, se verá una tabla como esta:

```
Ejercicio 2.1: TRUE
Ejercicio 2.2: TRUE
Ejercicio 2.3: TRUE
Ejercicio 2.4: TRUE
Ejercicio 2.5: TRUE
Ejercicio 2.6: TRUE
Ejercicio 2.7: TRUE
Ejercicio 2.8: TRUE
```

Ejercicio 3: Registros: Inversión de Pila de Libros

En este ejercicio, trabajaremos con pilas de registros. Aquí tomaremos de nuevo el rol de implementador del TAD pero esta vez almacenando información compleja (registros de tipo Libro) en lugar de tipos simples. Añadiremos una nueva funcionalidad que permita invertir una pila de libros. Para ello se proporcionan tres archivos Pascal:

- `uPilaLibros.pas`
- `uTLibro.pas`
- `pilas_ej3.pas`

`uPilaLibros.pas`

Este archivo contiene la definición de la pila de libros, que permite realizar operaciones básicas sobre una pila que almacena registros de tipo `TLibro`. Las operaciones disponibles son: inicializar la pila, agregar un libro a la pila, eliminar el libro de la cima de la pila, obtener el libro de la cima de la pila, comprobar si la pila está vacía e imprimir la pila. No es necesario modificar este archivo.

Deberías ser capaz de resolver el ejercicio sin necesidad de modificar este archivo.

Ejercicio 3.1 `uTLibro.pas`

Este archivo define el tipo de registro `TLibro`, que representará un libro en este ejercicio. El libro tiene los siguientes campos:

- `titulo`: string[100]
- `autor`: string[50]
- `ISBN`: string[20]
- `idioma`: string[20]
- `numPaginas`: integer

Además, incluye las dos siguientes funciones:

- **Asignar un libro a otro**

```
procedure assign(var e: TLibro; e2 : TLibro);
```

- **Convierte un libro a cadena**

```
function toString(e: TLibro): string;
```

Ejercicio 3.2 `pilas_ej3.pas`

En este archivo, se debe implementar la funcionalidad para invertir una pila de libros. Se proporcionan las siguientes funciones:

- **Invertir la pila**: Define un procedimiento `invertirPila` que invierte el orden de los libros en la pila.
  - **Entradas**:
    - Pila de libros.
  - **Salida**:
    - Modifica la pila para que los elementos estén en orden inverso.
  - **Ejemplos**:
    - `[libro1, libro2, libro3]` -> `[libro3, libro2, libro1]`
    - `[]` -> `[]`
    - `[libro1]` -> `[libro1]`

Comprobar el correcto funcionamiento de las funciones implementadas

En el programa principal, `pilas_ej3.pas`, se proporcionan ejemplos de cómo agregar libros a la pila, imprimir la pila original, invertirla y luego imprimir la pila invertida. Al ejecutar el programa, se verá una salida que muestra la pila original y la pila invertida, permitiendo verificar que la inversión se ha realizado correctamente.

```
Pila original:

(Wigetta en ED, Vegetta777 y Willyrex, 978-3-16-148410-2, Infantil, 2)
(Kika Superbruja y el libro de hechizos, Knister, 978-3-16-148410-1,
Alemán, 55)
(Don Quijote de Móstoles, Cervantes, 978-3-16-148410-0, Español, 9999)

Pila invertida:

(Don Quijote de Móstoles, Cervantes, 978-3-16-148410-0, Español, 9999)
(Kika Superbruja y el libro de hechizos, Knister, 978-3-16-148410-1,
Alemán, 55)
(Wigetta en ED, Vegetta777 y Willyrex, 978-3-16-148410-2, Infantil, 2)
```

Ejercicio 4: Palabras Palíndromas

Una palabra es un palíndromo si se lee igual de izquierda a derecha que de derecha a izquierda. Por ejemplo: `reconocer`, `oro` o `anilina`. En este ejercicio queremos verificar si una cierta palabra es un palíndromo utilizando pilas. Para ello, se proporcionan dos archivos Pascal:

- `uPilaChar.pas` (desarrollado en el Ejercicio 1)
- `pilas_ej4.pas`

El archivo `uPilaChar.pas` contiene la definición de la pila de caracteres, que se utilizará en este ejercicio. Por otro lado, el archivo `pilas_ej4.pas` es el programa principal que se utilizará para comprobar el correcto funcionamiento de las funciones implementadas.

Concretamente, se deberá implementar la siguiente función:

- **Verificar palíndromo**: Define una función `esPalindromo` que verifica si una palabra es un palíndromo o no.
  - **Entradas**:
    - Una cadena de caracteres que representa una palabra.
  - **Salida**:
    - Devuelve `true` si la palabra es un palíndromo, y `false` en caso contrario.
  - **Ejemplos**:
    - `anilina` -> `true`
    - `palindromo` -> `false`
    - `radar` -> `true`
    - `holita` -> `false`
    - `reconocer` -> `true`
    - `oso` -> `true`
    - `mameluco` -> `false`

Comprobar el correcto funcionamiento de las funciones implementadas

En el programa principal, `pilas_ej4.pas`, se proporcionan ejemplos de cómo verificar si una palabra es un palíndromo. Al ejecutar el programa, se verá una salida que muestra la palabra original y el resultado de la verificación, permitiendo verificar que la función `esPalindromo` se ha implementado correctamente.

```
Ejemplo #  Frase            Res = Esp    OK/ERROR
-----------------------------------------------------
Ejemplo 1: anilina          TRUE = TRUE  OK
Ejemplo 2: palindromo       FALSE = FALSE OK
Ejemplo 3: radar            TRUE = TRUE  OK
Ejemplo 4: holita           FALSE = FALSE OK
Ejemplo 5: reconocer        TRUE = TRUE  OK
Ejemplo 6: oso              TRUE = TRUE  OK
Ejemplo 7: mameluco         FALSE = FALSE OK
```

Ejercicio 5: Operaciones avanzadas con pilas con arrays

En este ejercicio, trabajaremos con operaciones avanzadas de pilas utilizando arrays. Para ello, se proporcionan dos archivos Pascal:

- `uPilaIntegerExtendedArray.pas`
- `pilas_ej5.pas`

Se deberá implementar las mismas funciones que en el Ejercicio 2, pero utilizando arrays en lugar de pilas dinámicas.

De manera similar, se deberá usar el archivo `uPilaIntegerExtendedArray.pas` para definir la pila de enteros extendida con arrays. Por otro lado, el archivo `pilas_ej5.pas` es el programa principal que se utilizará para comprobar el correcto funcionamiento de las funciones implementadas.