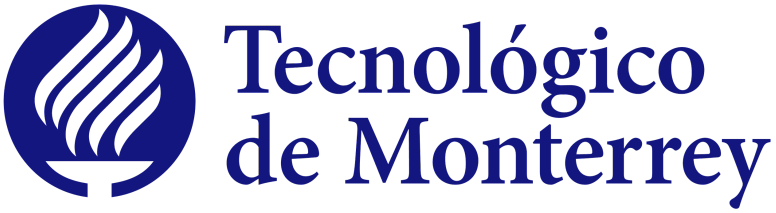
Instituto Tecnológico y de Estudios Superiores de Monterrey



Programación de estructuras de datos y algoritmos fundamentales (Gpo 608)

Profesor:
Eduardo López Benítez

Act 2.2 - Verificación de las funcionalidades de una estructura de datos lineal

Casos de prueba

Francisco Nicolás Jervis Hidalgo | A00835131

Con estos casos de prueba se busca probar el funcionamiento de varios métodos CRUD implementados en 2 tipos de estructuras de datos lineales. La primera estructura es una pila o stack la cual funciona a través de FIFO o first in first out lo que significa que datos nuevos solo pueden entrar desde el inicio de la estructura y similarmente sólo se pueden borrar los datos que se encuentran al inicio.

Por otro lado, la cola de prioridad tiene un funcionamiento de FILO o first in last out lo que significa que los datos son añadidos al inicio de la estructura pero solo se pueden sacar los datos que se encuentran al final de la misma. En este caso, la cola de prioridad tiene una funcionalidad adicional, está siendo que los datos tienen cierto nivel de prioridad que afectarán su posicionamiento en la cola. Con una prioridad más alta que el resto serán posicionados al final de la cola, similarmente, con una prioridad menor que todos, será posicionado al inicio de la cola. y con una prioridad la cual ya se encuentra en la cola este dato será posicionado antes que los que tienen la misma prioridad que él pero después que los datos que tienen una prioridad menor.

Para comprobar el funcionamiento de estas estructuras se utilizó dos funciones las cuales llenan las mismas con datos aleatorios. Estas funciones utilizan las funciones add para queue y push para stack lo que significa que los datos serán añadidos dependiendo de los parámetros especificados anteriormente.

```
void FillQueueInt(PriorityQueue<int>* list, int size) {
    srand((unsigned)time(NULL));
    int r,r2;
    for (int i = 0; i < size; i++) {
        r = rand() % 100;
        r2 = rand() % 5;
        list->add(r,r2);
    }
}
void FillStackInt(Stack<int>* list, int size) {
    srand((unsigned)time(NULL));
    int r;
    for (int i = 0; i < size; i++) {
        r = rand() % 100;
        list->Push(r);
    }
}
```

Después de esto se imprimen todos los datos en cada estructura

```
pQueue->PrintAll();
stack->PrintAll();
```

```
Listas originales:
 52 p(0) 71 p(0) 67 p(0) 31 p(3) 25 p(3) 88 p(3) 30 p(4) 82 p(4) 69 p(4) 51 p(4)

*5
*52
*55
*71
*4
*51
*20
*67
*33
*88
```

La cola de prioridad está impresa de manera horizontal con la prioridad de cada dato en paréntesis mientras que la pila fue impresa de manera vertical.

Para comprobar la función create de cada estructura se añadieron 3 datos nuevos a cada una y se volvieron a imprimir.

```
cout << "Create: \n";
pQueue->add(7, 0);
pQueue->add(91, 4);
pQueue->add(53, 2);

stack->Push(62);
stack->Push(19);
stack->Push(31);

pQueue->PrintAll();
stack->PrintAll();
```

```
Create:
 7 p(0) 52 p(0) 71 p(0) 67 p(0) 53 p(2) 31 p(3) 25 p(3) 88 p(3) 91 p(4) 30 p(4) 82 p(4) 69 p(4) 51 p(4)

*31
*19
*62
*5
*52
*55
*71
*4
*51
*20
*67
*33
*88
```

Como se puede observar, los datos fueron almacenados en las estructuras dependiendo de su funcionalidad, en el caso de la pila todos los datos fueron al principio y en caso de la cola estos fueron posicionados dependiendo de su prioridad.

Después de esto se comprobó la funcionalidad de read, una de estas funciones ya fue utilizada, la cual imprime toda la estructura, pero adicionalmente cada estructura tiene una función peek que dependiendo de su funcionamiento imprime diferentes datos. para el stack el dato al inicio y para la cola el dato al final.

```
cout << "Read:\n";

cout << "Tail:" << pQueue->Peek()<<"\n\n";

cout << "Top:" << stack->Peek() << "\n\n";

cout << "Delete:\n";
```

```
Read:
Tail:51

Top:31
```

Al comparar con lo impreso anteriormente se puede comprobar que esto es correcto.

Finalmente, se debe comprobar las funciones de delete las cuales borran un elemento dependiendo de la funcionalidad de la estructura.

```
pQueue->remove();
stack->Pop();

pQueue->PrintAll();
stack->PrintAll();
```

```
Delete:
 7 p(0) 52 p(0) 71 p(0) 67 p(0) 53 p(2) 31 p(3) 25 p(3) 88 p(3) 91 p(4) 30 p(4) 82 p(4) 69 p(4)

*19
*62
*5
*52
*55
*71
*4
*51
*20
*67
*33
*88
```

Como se puede observar, los elementos fueron eliminados adecuadamente.

Output completo:

```
Listas originales:
 52 p(0) 71 p(0) 67 p(0) 31 p(3) 25 p(3) 88 p(3) 30 p(4) 82 p(4) 69 p(4) 51 p(4)

*5
*52
*55
*71
*4
*51
*20
*67
*33
*88

Create:
 7 p(0) 52 p(0) 71 p(0) 67 p(0) 53 p(2) 31 p(3) 25 p(3) 88 p(3) 91 p(4) 30 p(4) 82 p(4) 69 p(4) 51 p(4)

*31
*19
*62
*5
*52
*55
*71
*4
*51
*20
*67
*33
*88

Read:
Tail:51

Top:31

Delete:
 7 p(0) 52 p(0) 71 p(0) 67 p(0) 53 p(2) 31 p(3) 25 p(3) 88 p(3) 91 p(4) 30 p(4) 82 p(4) 69 p(4)

*19
*62
*5
*52
*55
*71
*4
*51
*20
*67
*33
*88
```