Práctica 6: Cola circular y cola doble

Estructuras de Datos y Algoritmos I

Autor: José Mauricio Matamoros de Maria y Campos

1. Objetivo

El alumno revisará las definiciones, características, procedimientos y ejemplos de las estructuras lineales *cola circular* y *cola doble* para poder comprenderlas a fin de ser capaz de implementarlas.

2. Material

Se asume que el alumno cuenta con un una computadora con arquitectura Intel x86 o compatible, sistema operativo Linux/Unix, y compilador `gcc` instalado.

3. Instrucciones

1. Lea detenidamente los antecedentes teóricos sobre la construcción de colas circulares y colas dobles Sección 4.
2. Realice cada una de las actividades detalladas en la Sección 5 apoyándose en el código de ejemplo del Apéndice A y responda las preguntas de las mismas.

4. Antecedentes

4.1. Cola circular

Una cola circular es una modificación a la estructura de datos lineal cola (*queue*) en la cual el último elemento apunta al primer elemento en lugar de a *nulo*. Así, una cola circular funciona igual que si la operación `dequeue` llamara internamente a `enqueue` volviendo a colocar al elemento retirado en la cola, pero de manera más eficiente, por lo que los elementos nunca son realmente desencolados.

Es importante recalcar que una cola circular sigue el mismo principio que un buffer circular, con diferencia de que este último es de tamaño fijo, mientras que una cola circular puede expandirse dinámicamente.

Nota sobre variantes y autores

El concepto teórico de *cola* (*queue*) se refiere a una estructura de datos tipo *FIFO* (***F**irst **I**n, **F**irst **O**ut* o *primero en entrar, primero en salir*) con capacidad infinita y en la cual todas las operaciones se ejecutan en tiempo constante $O(1)$. Diferentes implementaciones como el uso de arreglos, apuntadores, o búferes circulares pertenecen al campo de la ingeniería y las definiciones de cada tipo o variante pueden variar según el autor.

4.2. Deque

Una cola doble o deque (*double-ended queue*) es una lista lineal en la cual todas las operaciones de inserción y eliminación se realizan en los extremos. Por lo tanto, una cola doble es más general que una pila o una cola y tiene algunas propiedades en común con una baraja de cartas. Por otro lado, existen colas dobles restringidas en las que las operaciones de inserción o eliminación se permiten en sólo uno de los extremos.

Cuando se habla de colas dobles, los elementos a los que uno se refiere comúnmente son los extremos izquierdo y derecho en lugar de cabeza y rabo (*head* y *tail*), aunque los conceptos de *base (top)* y *tope (bottom)*, o *frente (front)*

y *trás (back)*, a veces se aplican a colas dobles que se utilizan como pilas o colas. Es decir, no existe convención ni estándar sobre si la parte superior, inferior, delantera y trasera deben aparecer a la izquierda oa la derecha. Por lo tanto, resulta fácil utilizar una rica variedad de palabras descriptivas del idioma inglés en los algoritmos.

La implementación de una cola doble es trivial pues generaliza las operaciones de encolado y desencolado para ambos lados, siendo las operaciones estándar de C++[1]:

- `push_front`
- `push_back`
- `pop_front`
- `pop_back`

5. Desarrollo de la práctica

Lea cuidadosamente los problemas descritos a continuación e implemente la estructura de datos necesaria para hacer que el programa propuesto funcione.
Hint: Utilice las implementaciones de la práctica anterior.

5.1. Actividad 1: Semáforo

Se desea implementar un controlador genérico para una luz de semáforo de N fases. Las señales del semáforo (símbolo, color y duración) están codificadas como números enteros que se irán desplegando en orden cíclico.

Complete el programa de la Apéndice A.2 implementando la estructura de datos correspondiente y ejecute el programa
¿Qué salida produce el programa? [1 punto]: _____

¿Qué estrucura de datos usó? Explique [1 punto]: _____

¿Qué modificaciones realizó a la estructura de datos tipo cola para implementar la estructura de datos empleada? Anote y explique su código a continuación [3 puntos]:

[1] http://www.cplusplus.com/reference/deque/deque/

5.2. Actividad 2: Sistema de turnos

Un banco desea implementar un sistema de turnos inteligente para atender de manera eficiente y ordenada a sus clientes, el cual deberá operar de la siguiente manera:

i) Todo cliente deberá formarse en una unifila al llegar.
ii) En general, los clientes son atendidos en el orden en el que llegan.
iii) Si un cliente precisa de llenar un formato, este dejará la ventanilla libre.
iv) Tras llenar un formato, un cliente no volverá a formarse en la parte trasera de fila, sino que esperará al frente a que una ventanilla se desocupe

Complete el programa de la Apéndice A.2 implementando la estructura de datos correspondiente y ejecute el programa

¿Qué salida produce el programa? [1 punto]: _____

¿Qué estrucura de datos usó? Explique [1 punto]: _____

¿Qué modificaciones realizó a la estructura de datos tipo cola para implementar la estructura de datos empleada? Anote y explique su código a continuación [3 puntos]:

A. Código de ejemplo

A.1. Archivo `src/Makefile`

src/Makefile

```
CC       = gcc
CFLAGS   = -Wall -O0
SILENT   = @
PROGRAMS = sem bank
.PHONY: all clean $(PROGRAMS)

all: $(PROGRAMS)

$(PROGRAMS): %: %.c
  $(SILENT) $(CC) $(CFLAGS) -o $@ $<

clean:
  rm -f *.o
```

A.2. Archivo `src/sem.c`

src/sem.c

```c
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>

// Estructuras de datos
typedef int queue;  // Reemplazar int con el nombre de la estrucutra

// Prototipos
void enqueue(queue* q, int sym);
int dequeue(queue* q);

void load_symbols(queue* q);
void display(int sym);

int main(void){
  // 1. Inicializar estructura de datos (queue)
  queue *q = NULL;

  // 2. Cargar simbolos
  load_symbols(q);

  // 3. Ejecutar 10 ciclos de semaforo
  for(size_t i = 0; i < 10; ++i)
    display(dequeue(q));

  return 0;
}

void load_symbols(queue* q){
  int symbols[5] = {
    8 << 16 | 'R' << 8 | 'X',
    2 << 16 | 'Y' << 8 | 'O',
    4 << 16 | 'G' << 8 | 'L',
    4 << 16 | 'G' << 8 | 'R',
    8 << 16 | 'G' << 8 | 'O',
  };
  for(size_t i = 0; i < 5; ++i)
    enqueue(q, symbols[i]);
}

void display(int sym){
  char *color;
  int glyph = sym & 0xFF;
  int delay = (sym >> 16)* 250000;
  switch((sym >> 8) & 0xFF){
    case 'R': color = "\e[1;31m"; break;
    case 'G': color = "\e[1;32m"; break;
    case 'Y': color = "\e[1;33m"; break;
    case 'B': color = "\e[1;34m"; break;
    case 'P': color = "\e[1;35m"; break;
    case 'C': color = "\e[1;36m"; break;
    case 'W': color = "\e[1;37m"; break;
    default: color = "\e[0m"; break;
  }
  printf(" %s %c\e[0m\n", color, glyph);
  usleep(delay);
}
```

A.3. Archivo `src/bank.c`

src/bank.c

```c
#define CASHIERS 4
#define LOBBY 20

#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>

// Estructuras de datos
typedef int queue;  // Reemplazar int con el nombre de la estrucutra

// Prototipos
void enqueue(queue* q, int cli); // Forma ATRáS de la cola
int dequeue(queue* q); // Saca del frente de la cola
void push(queue* q, int cli); // Forma EN FRENTE de la cola
int is_empty(queue* q);
int is_full(queue* q);

int gen_client(int count);
void populate_cashiers(int* cashiers, queue* q);
void do_work(int* cashiers, int* lobby, queue* q);
void fill_forms(int* lobby, queue* q);

int main(void){
  int new_client;
  int tot_clients = 0;
  int* cashiers = (int*)calloc(CASHIERS, sizeof(int));
  int* lobby = (int*)calloc(LOBBY, sizeof(int));
  // Init random seed
  srand(69);

  // 1. Inicializar estructuras de datos (queue)
  queue *q = NULL; // La fila

  // 2. Ejecutar hasta atender 30 clientes
  while((tot_clients < 30) || !is_empty(q)){
    // Generar cliente
    new_client = gen_client(tot_clients++);
    // Si un cliente llega y la cola esta llena, se va
    if(!is_full(q)){
      printf("Cliente %d llega al banco y se forma.\n", new_client >> 16);
      enqueue(q, new_client);
    }
    else printf("Banco lleno. Cliente %d se va.\n", new_client >> 16);

    // Los clientes pasan a las cajas vacias
    populate_cashiers(cashiers, q);

    // Las cajas atienden a los clientes
    // y los clientes llenan sus formatos
    do_work(cashiers, lobby, q);
  }
  printf("Fin de jornada.\n");
  return 0;
  random();
}

int gen_client(int count){
  int cli_time = 1 + random() % 5;
  int inc = (random() % 100) < 20 ? 1 + random() % 3 : 0;
  return (count << 16) + (cli_time << 8) + inc;
}

void populate_cashiers(int* cashiers, queue* q){
  for(int i = 0; i < CASHIERS; ++i){
```

```
65      if ((cashiers[i] != 0) || is_empty(q))
66        continue;
67      cashiers[i] = dequeue(q);
68      printf("Cliente %d pasa a la caja %d.\n", cashiers[i] >> 16, i);
69      usleep(125000);
70    }
71  }
72
73  void do_work(int* cashiers, int* lobby, queue* q){
74    int ix = 0;
75    int cli_time;
76    // Los clientes llenan sus formatos
77    fill_forms(lobby, q);
78    // Las cajas hacen su trabajo
79    for(int i = 0; i < CASHIERS; ++i){
80      if(cashiers[i] == 0) continue;
81      usleep(125000);
82      if(cashiers[i] & 0xFF){
83        while(lobby[ix] != 0) ++ix;
84        printf("Cliente %d en caja %d tiene que llenar formato.\n", cashiers[i] >> 16, i);
85        lobby[ix++] = cashiers[i];
86        cashiers[i] = 0;
87        continue;
88      }
89      cli_time = ((cashiers[i] & 0xFF00) >> 8) -1;
90      if(cli_time <= 0){
91        printf("Cliente %d en caja %d se retira.\n", cashiers[i] >> 16, i);
92        cashiers[i] = 0;
93        continue;
94      }
95      cashiers[i] = (cashiers[i] & 0xFF00FF) | (cli_time << 8);
96    }
97  }
98
99  void fill_forms(int* lobby, queue* q){
100   for(int i = 0; i < LOBBY; ++i){
101     if(lobby[i] == 0) continue;
102     if((lobby[i] & 0xFF) == 0){
103       push(q, lobby[i]);
104       printf("Cliente %d termina de llenar formato y se mete en la fila.\n", lobby[i] >> 16)
      ;
105       lobby[i] = 0;
106     }
107     else --lobby[i];
108   }
109 }
```

B. Reporte Escrito

El reporte de la práctica deberá ser entregada en un archivo en formato `PDF` siguiendo las siguientes especificaciones:

- La primera página del documento deberá ser la carátula oficial para prácticas de laboratorio disponible en `lcp02.fi-b.unam.mx/`
- El nombre del documento `PDF` deberá ser `nn-XXXX-L06.pdf`, donde:
  - `nn` es el número de lista del alumno a dos dígitos forzosos (ej. 01, 02, etc.).
  - `XXXX` corresponden a las dos primeras letras del apellido paterno seguidas de la primera letra del apellido materno y la primera letra del nombre, en mayúsculas y evitando cacofonías; es decir, los cuatro primeros caracteres de su RFC o CURP.
- El reporte consiste en un documento de redacción propia donde el alumno explica de forma concisa y a detalle las actividades realizadas en la práctica, y reportando los resultados obtenidos.
- La longitud del reporte no deberá exceder las 3 páginas, sin contar la carátula.
- El reporte deberá seguir todos los lineamientos para documentos escritos establecidos al inicio del curso.
- Todas las referencias deberán estar debidamente citadas.

IMPORTANTE: No se aceptan archivos en otros formatos ni con nombres distintos a los especificados.