Laboration 5

Mål: Du ska på att använda containerklasser i C++. Du ska också implementera egna containerklasser med hjälp av pekare och fält.

Läsanvisningar

Läs kap 12–13 i läroboken. Läs också föreläsningsbilder från de föreläsningar, som behandlar motsvarande avsnitt. Dessa finns på kursens hemsida.

Förberedelser

Läs igenom den inledande texten under rubriken ”Uppgifter” nedan och lös uppgift U1 eller U2. Läs igenom och sätt dig in i uppgifterna U3-U6. Uppgift U4 och U5 är frivilliga.

Uppgifter

I denna uppgift ska du

U1. Denna uppgift går ut på att simulera en bankomatkö med användning av en container av slaget queue. Kön ska innehålla kunder av följande slag:

```
class Kund {
  int atid; // ankomsttid
  int btid; // betjäningstid
public:
  // medlemsfunktioner
}
```

Förutom konstruktorer (som initierar datamedlemmarna på lämpligt sätt) behöver `Kund`-klassen medlemsfunktioner för att avläsa ankomsttid och betjäningstid. Betjäningstiden slumpas fram (1-4 minuter med lika stor sannolikhet). Det är lämpligt (men ej nödvändigt) att utföra detta i en konstruktor. Själva simuleringen går till på så sätt att användaren får ange hur många timmar som ska simuleras och det genomsnittliga antalet kunder/timme (högst 60). Sedan används en loop där varje varv får motsvara 1 minut, varvid man kan göra på följande sätt:

1. Slumpa fram nya kunder (så att det genomsnittliga antalet kunder/timme uppfylls). Under C++11 kan `std::poisson_distribution` som finns i biblioteket `<random>` användas.
2. Varje ny kund som anländ (om någon) placeras i kön.
3. Om bankomaten är ledig börjar man betjäna kunden som står främst i kön (om det finns någon kund att betjäna).
4. Minska återstående betjäningstid med 1 för den kund som betjänas.

Programmet ska hålla reda på totala antalet betjänade kunder och total kötid. För att det ska vara möjligt att beräkna den genomsnittliga kölängden när själva simuleringen är klar behöver man dessutom varje minut avläsa och summera den aktuella kölängden (summan sparas i en särskild variabel för ackumulerad kölängd). När simuleringen är klar redovisas statistik.

U2. Som föregående uppgift, men presentera resultatet grafiskt. Möjliga utvidgningar:

1. Spara kölängderna för varje minut och presentera dessa i ett stapeldiagram över genomsnittlig kölängd under varje femminutersintervall.

2. Visa grafiskt hur kön förändras under simulerad tid. Använd t.ex. en `wxSpinCtrl` för att styra hastigheten på simuleringen. Låt ”huvudloopen” ligga i grafiken och anropa modellen för uppdatering en gång varje simulerad minut.

U3. Skriv ett program som framslumpar positiva heltal (0-9999) och insätter dessa först i en enkellänkad lista. Skriv ut listan och bestäm största talet (låt funktioner utföra jobbet).

U4. Skriv en funktion som söker efter ett tal i en dubbellänkad lista av det slag som beskrivs i C++ direkt. Funktionen ska ha listan och det sökta talet som parametrar och returnera en pekare till noden med talet om det hittades och en NULL-pekare annars.

U5. Läroboken implementerar en stack med en länkad lista, men det går även utmärkt att utnyttja ett fält till detta. Implementera en sådan stack med utgångspunkt från nedanstående definition. Lägg märke till att stackens storlek ska kunna bestämmas när stacken skapas och att pop fungerar på annat sätt än bokens version.

```
#include <stdexcept>
class Stack {
private:
  int maxSize; // Stackens maximala storlek
  int *data;   // Data
  int num;     // Antalet element i stacken
public:
  Stack(int max=100);
  ~Stack();
  void clear();
  bool empty() const;
  bool full() const ;
  int pop(); // Hämta+avlägsna element, throw (length_error)
  void push(int value);  throw (length_error)
};
```

U6. Det är även möjligt att implementera köer med fält, fast det är knivigare. Problemet är att i en kö sker operationerna i båda ändarna. Ett sätt att lösa detta är att använda en slags rullande buffert (cirkulär kö), där man börjar om från andra änden när man nått buffertens slut. Implementera en sådan kö med utgångspunkt från följande definitioner:

```
#include <stdexcept>
class Queue {
private:
  int maxSize; // Köns maximala storlek
  int *data;   // Data
  int head;    // index till nästa pos att läsa från
  int tail;    // index till nästa pos att skriva till
public:
  Queue(int max=100);
  ~Queue();
  void clear();
  bool empty() const;
  bool full() const;
  int dequeue(); //Hämta+avlägsna elem, throw (length_error)
  void enqueue(int value); // throw (length_error)
  int length();
};
```

Tips: Det kan vara lite trixigt att få ihop det hela, särskilt gäller detta kontrollen om kön är full. Enklast hanteras detta med en räknare som håller reda på antalet element i kön (det är tillåtet att komplettera specifikationen med en sådan variabel).