ЛАБОРАТОРНА РОБОТА №15

Тема: **Стек та черга**
Мета: **Ознайомитись з основами роботи зі стеком та чергою**
Час: 2 год.

Виконання роботи

- Надати викладачу, виконане завдання для самопідготовки в п. 4.2.
- Вивчити теоретичні відомості.
- Відповісти тестові завдання.
- Виконати самостійну роботу.

Завдання для самопідготовки

В процесі підготовки до заняття студент у обов'язковому порядку повинен виконати наступні завдання:

а) За допомогою конспекту лекцій і рекомендованої літератури розглянути суть таких питань:

1) Поняття, способи створення та обробки стеку
2) Поняття, способи створення та обробки черги

б) Занести в звіт такі дані:

1) номер практичної роботи;
2) тему і мету роботи;
3) короткий конспект основних теоретичних відомостей.

Теоретичні відомості

СТЕК

Стеком називається упорядкований набір елементів, в якому розміщення нових і видалення існуючих відбувається з одного кінця, званого вершиною.

Дисципліна обслуговування - це сукупність правил (впорядкування і алгоритм) обслуговування елементів динамічної структури даних.

Залежно від дисципліни обслуговування розрізняють ті чи інші структури динамічних даних.

Принцип роботи стека порівнюють зі стопкою аркушів паперу: щоб взяти другий зверху, потрібно зняти верхній.

У стеці реалізується дисципліна обслуговування LIFO:

- LAST - останній
- INPUT - увійшов
- FIRST - перший
- OUTPUT - вийшов

Розрізняють апаратний і програмний стек.

Апаратний стек використовується для зберігання адрес повернення з функцій і їх аргументів.

Програмний стек - це призначена для користувача модель (структура) даних.

Операції для роботи зі стеком

Над стеком реалізовані наступні операції:

- ініціалізація стека `init (s)`, де `s` - стек
- приміщення елемента в стек `push (s, i)`, де `s` - стек, `i` - поміщається елемент;
- видалення елемента з стека `i = pop (s);`
- визначення верхнього елементу без його видалення `i = stkTop (s)`, яка еквівалентна операціями `i = pop (s); push (s, i);`
- отримання вершини стека (кількості елементів) `i = gettop (s)`, де `s` - стек
- друк стека `stkPrint (s)`, де `s` - стек
- визначення порожнечі стека `isempty (s)`
- повертає `true` якщо стек порожній і `false` в іншому випадку.

Способи реалізації стека

Існує кілька способів реалізації стека:

- за допомогою одновимірного масиву;
- за допомогою пов'язаного списку;
- за допомогою класу об'єктно-орієнтованого програмування.

Приклад реалізації стека

Стек можна реалізувати у вигляді такої структури:

```
#define NMAX 100
struct stack {
  float elem [NMAX];
  int top;
};
```

`NMAX` - максимальна кількість елементів в стеку;

`elem` - масив з `NMAX` чисел типу `float`, призначений для зберігання елементів стека;

`top` - індекс елемента, що знаходиться в вершині стека.

Ініціалізація стека

Індекс елемента, що знаходиться в вершині стека, дорівнює 0.

```
void init (struct stack * stk) {
  stk-> top = 0;
}
Приміщення елемента в стек
void push (struct stack * stk, float f) {
  if (stk-> top <NMAX) {
    stk-> elem [stk-> top] = f;
    stk-> top ++;
  } else
    printf ( "Стек сповнений, кількість 
елементів:% d! \ n", stk-> top);
```

```
}
```

Видалення елемента з стека

```
float pop (struct stack * stk) {
  float elem;
  if ((stk-> top)> 0) {
    stk-> top--;
    elem = stk-> elem [stk-> top];
    return (elem);
  } Else {
    printf ( "Стек порожній! \ n");
    return (0);
  }
}
```

Витяг вершини стека

```
float sktTop (struct stack * stk) {
  if ((stk-> top)> 0) {
    return (stk-> elem [stk-> top-1]);
  } Else {
    printf ( "Стек порожній! \ n");
    return (0);
  }
}
```

Отримання верхнього елементу стека без його видалення

```
int gettop (struct stack * stk) {
 return (stk-> top);}
Визначення порожнечі стека
int isempty (struct stack * stk) {
  if ((stk-> top) == 0) return (1);
  else return (0);
}
```

Повернення елементів стека

```
void stkPrint (struct stack * stk) {
  int i;
  i = stk-> top;
  if (isempty (stk) == 1) return;
  do {
    i--;
    printf ( "% f \ n", stk-> elem [i]);
  } While (i> 0);
}
```

Черга

Чергою називається упорядкований набір елементів, які можуть вилучатися з її початку і поміщатися в її кінець.

Черга організована, на відміну від стека, згідно дисципліни обслуговування FIFO:

- FIRST - перший
- INPUT - увійшов
- FIRST - перший
- OUTPUT - вийшов

Черга в програмуванні використовується, як і в реальному житті, коли потрібно зробити якісь дії в порядку їх надходження, виконавши їх послідовно. Прикладом може служити організація подій в Windows. Коли користувач робить якась агресивна дія на додаток, то в додатку не викликається відповідна процедура (адже в цей момент додаток може здійснювати інші дії), а йому надсилається повідомлення, що містить інформацію про скоєний дії, це повідомлення ставиться в чергу, і тільки коли будуть оброблені повідомлення, що прийшли раніше, додаток виконає необхідну дію.

Існує кілька способів реалізації черги:

- за допомогою одновимірного масиву;
- за допомогою пов'язаного списку;
- за допомогою класу об'єктно-орієнтованого програмування.

Найпростіші операції з чергою:

- `init ()` ініціалізація черги.
- `insert (q, x)` - приміщення елемента x в кінець черги `q` (`q` - покажчик на чергу);
- `x = remove (q)` - видалення елемента x з черги q;
- `isempty (q)` - повертає `true (1)`, якщо чергу порожня і `false (0)` в іншому випадку;
- `print (q)` - висновок елементів черги q.

Розглянемо реалізацію черзі на базі масиву. Використовуємо масив і дві змінні:

- `frnt` - позиція першого елемента в черзі;
- `rear` - позиція останнього елемента в черзі

Спочатку `frnt = 1` і `rear = 0`.

Черга порожня, якщо `rear <frnt`.

Число елементів в черзі можна визначити як

```
n = rear-frnt + 1
#define QMAX 100
struct queue {
  int qu [QMAX];
  int rear, frnt;
};
```

Ініціалізація черги

```
void init (struct queue * q) {
  q-> frnt = 1;
  q-> rear = 0;
  return;
}
```

Додавання елемента в чергу

```
void insert (struct queue * q, int x) {
  if (q-> rear <QMAX-1) {
    q-> rear ++;
    q-> qu [q-> rear] = x;
  }
  else
    printf ( "Черга повна! \ n");
  return;
}
```

Перевірка порожнечі черзі

```
int isempty (struct queue * q) {
  if (q-> rear <q-> frnt) return (1);
  else return (0);
}
```

Повернення елементів черги

```
void print (struct queue * q) {
  int h;
  if (isempty (q) == 1) {
    printf ( "Черга порожня! \ n");
    return;
  }
  for (h = q-> frnt; h <= q-> rear; h ++)
    printf ( "% d", q-> qu [h]);
  return;
}
```

Видалення елемента з черги

```
int remove (struct queue * q) {
  int x;
  if (isempty (q) == 1) {
    printf ( "Черга порожня! \ n");
    return (0);
  }
  x = q-> qu [q-> frnt];
  q-> frnt ++;
  return (x);
```

}

Самостійна робота

1. У таблиці A розміру N за один перегляд необхідно кожен елемент замінити на найближчий наступний за ним елемент, який більше його. Якщо такого елемента немає, то замінити його на нуль. Можна використовувати додаткову пам'ять.

   Пояснення: *Необхідно організувати стек для позицій елементів, які претендують бути великими. Для кожного поточного елемента виштовхувати з стека все позиції, на яких стоять елементи менше поточного і замінити їх поточним. Потім позицію поточного елемента помістити в стек. Після перегляду всіх елементів в стеку стоятимуть позиції елементів, які треба замінити на нуль.*

2. Дана кінцева послідовність, що складається з лівих і правих дужок різноманітним заданих типів. Як визначити, чи можна додати в неї цифри і знаки арифметичних дій так, щоб вийшло правильне арифметичне вираз.

   Пояснення: *Ідея рішення грунтується на використанні стека. Зчитуючи вхідну послідовність дужка за дужкою діє таким чином.*

   1. Якщо чергова дужка - відкриває, то поміщаємо її в стек.

   2. Якщо чергова дужка - закриває, то аналізуємо дужку, що стоїть в вершині стека. Можливо кілька ситуацій:

   а) відкриває дужка відповідає черговий закриває. В цьому випадку вона виштовхується з стека, а процес з нових вхідних дужкою.

   б) відкриває дужка не відповідає черговий закриває або стек порожній. У цьому випадку неможливо отримати правильне арифметичне вираз.

   Коли все дужки вхідного рядка оброблені, можливі 2 ситуації.

   1. Стек порожній. В цьому випадку можна отримати правильне арифметичне вираз.

   2. стік не порожній. У цьому випадку неможливо отримати правильне арифметичне вираз.

3. Дан набір з 10 чисел. Створити дві черги: перша повинна містити числа з вихідного набору з непарними номерами (1, 3, ..., 9), а друга - з парними (2, 4, ..., 10); порядок чисел в кожній черзі повинен збігатися з порядком чисел в початковому наборі..

4. По колу розміщено N монет гербами вгору і M монет гербами вниз. Обходячи коло по ходу годинникової стрілки, перевертає кожну S -тую монету. У перший раз рахунок починається з герба. У якому порядку

треба розставити монети, щоб після K ходів стало L монет, що лежать гербами вгору.

Пояснення. *Монети лежать на N + M позиціях. Пронумеруємо ці позиції по порядку по контуру від 1 до N + M.*

Заведемо масив A з N + M осередків. Спочатку все осередки нульові. Починаючи рахунок від першого осередку, будемо робити хід - відраховувати S осередків (вважаємо, що за N + M-им елементом слід безпосередньо 1-ий елемент масиву) і замінювати в цьому осередку число і на число 1-і (тобто 0 на 1, а 1 на 0). Після k-того ходу зупинимося.

Розглянемо ситуацію, що виникла. Після кожного ходу перевертається одна монета, при цьому різниця кількості монет, що лежать гербами вгору і кількості монет, що лежать гербами вниз або збільшується, або зменшується на 2. Наприклад, якщо перевертається монета, що лежить гербом вгору, то при цьому збільшується на 1 кількість монет гербом вниз і зменшується на 1 кількість монет гербом вгору.

Припустимо, що після k ходів в масиві A стало p одиниць тобто p монет, в порівнянні з початковим становищем, будуть перевернуті після k-ого ходу.

У разі, якщо L> = N, то (L-N) монет, які лежали гербами вниз, повинні після k-ого ходу бути перевернуті гербами вгору. (Якщо p <(L-N), то це, очевидно, неможливо зробити). Серед решти p- (L-N) перевернутих монет повинна бути половина гербами вгору і половина - вниз, щоб при перевороті сумарне число монет гербами вгору і вниз не змінилося. Отже, число p- (L-N) має бути парним, інакше умові завдання задовольнити не можна. Нехай p- (L-N) = 2v. Повинно бути, очевидно, v <= N, v + (L-N) <= M.

Отже, в разі L> = N, якщо не виконується хоча б одна з нерівностей

p- (L-N) = 2v> = 0,

v <= N,

v + (L-N) <= M,

то перетворення початкової конфігурації в кінцеву неможливо.

Інакше на (L-N) місць, позначених в масиві A одиницями, виставляємо монети гербами вниз. На решту 2v = p- (L-N) помічених одиницями позицій кладемо v монет гербами вниз і v гербами вгору в довільному порядку. На інші позиції кладемо залишилися монети знову ж в довільному порядку, щоб в цілому було N монет гербами вгору і M - гербами вниз.

Вимоги до оформлення звіту

Звіт повинен містити:

– Короткий конспект теоретичних відомостей;
– Результати виконаних дій.