Міністерство освіти і науки України
Національний технічний університет України
«Київський політехнічний інститут імені Ігоря Сікорського»
Факультет інформатики та обчислювальної техніки
Кафедра обчислювальної техніки

Лабораторна робота №2

з дисципліни
«Алгоритми і структури даних»

Виконав:
студент групи ІМ-31
Литвиненко Сергій Андрійович
номер у списку групи: 12

Перевірила:
Молчанова А. А.

Київ 2024

Завдання

1. Створити список з **n** (**n** > 0) елементів (**n** вводиться з клавіатури), якщо інша кількість елементів не вказана у конкретному завданні за варіантом.

2. Тип ключів (інформаційних полів) задано за варіантом.

3. Вид списку (черга, стек, дек, прямий однозв'язний лінійний список, обернений однозв'язний лінійний список, двозв'язний лінійний список, однозв'язний кільцевий список, двозв'язний кільцевий список) вибрати самостійно з метою найбільш доцільного розв'язку поставленої за варіантом задачі.

4. Створити функції (або процедури) для роботи зі списком (для створення, обробки, додавання чи видалення елементів, виводу даних зі списку в консоль, звільнення пам'яті тощо).

5. Значення елементів списку взяти самостійно такими, щоб можна було продемонструвати коректність роботи алгоритму програми. Введення значень елементів списку можна виконати довільним способом (випадкові числа, формування значень за формулою, введення з файлу чи з клавіатури).

6. Виконати над створеним списком дії, вказані за варіантом, та коректне звільнення пам'яті списку.

7. **При виконанні заданих дій, виводі значень елементів та звільненні пам'яті списку вважати, що довжина списку (кількість елементів) <u>невідома</u> на момент виконання цих дій.** Тобто, не дозволяється зберігати довжину списку як константу, змінну чи додаткове поле.

При проєктуванні програм слід врахувати наступне:

1. при виконанні завдання кількість операцій (зокрема, операцій читання й запису) має бути мінімізованою, а також максимально мають використовуватися властивості списків;

2. повторювані частини алгоритму необхідно оформити у вигляді процедур або функцій (для створення, обробки, виведення та звільнення пам'яті списків) з передачею списку за допомогою параметра(ів).

3. у таких видів списків, як черга, стек, дек функції для роботи зі списком мають забезпечувати роботу зі списком, відповідну тому чи іншому виду списку (наприклад, не можна додавати нові елементи всередину черги);

4. програми мають бути написані мовою програмування С.

Варіант 12

Ключами елементів списку є цілі числа. Обчислити значення виразу: $(a_1 + a_2 + 2a_n)(a_2 + a_3 + 2a_{n-1}) \dots (a_{n-1} + a_n + 2a_2)$, де a_1 – і-тий елемент списку.

Текст програми

1. Реалізація на основі двохзв'язного циклічного списку:

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

typedef struct linkedList {
  int data;
  struct linkedList* prev;
  struct linkedList* next;
} linkedList;

linkedList* listInit() {
  return NULL;
}

linkedList* listPush(linkedList* list, int data) {
  linkedList* node = malloc(sizeof(linkedList));
  if (node == NULL) return NULL;
  if (list == NULL) {
    *node = (linkedList){ data, node, node };
    return node;
  }
  linkedList* tail = list->prev;
  *node = (linkedList){ data, tail, list };
  tail->next = node;
  list->prev = node;
  return node;
}

void listDestroy(linkedList* list) {
  linkedList* tail = list->prev;
  linkedList* prev = NULL;
  while (tail != list) {
    prev = tail->prev;
```

```
    free(tail);
    tail = prev;
  }
  free(list);
}

int calculate(linkedList* list) {
  linkedList* tail = list->prev;
  linkedList* head = list;
  int result = 1;
  while (tail != list) {
    result *= (tail->data + tail->prev->data + 2 * head->data);
    tail = tail->prev;
    head = head->next;
  }
  return result;
}

int main(int argc, char const *argv[]) {
  linkedList* list = listInit();
  int length, data, success = 1;
  while (1) {
    printf("Enter the count of elements you are going to enter: ");
    scanf("%d", &length);
    if (length > 1) break;
    printf("The value must be greater than 1\n");
  }
  for (int i = 1; i <= length; i++) {
    printf("Enter the element #%d: ", i);
    scanf("%d", &data);
    list = listPush(list, data);
    if (list == NULL) {
      printf("Error: There is not enough memory");
      success = 0;
```

```
      break;
    }
  }
  if (success) {
    printf("Result: %d\n", calculate(list));
  }
  listDestroy(list);
  return 0;
}
```

2. Реалізація на основі двохзв’язного списку з заголовком:

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

typedef struct node {
  int data;
  struct node* previos;
  struct node* next;
} node;

typedef struct {
  struct node* first;
  struct node* last;
} linkedList;

linkedList* listInit() {
  linkedList* list = malloc(sizeof(linkedList));
  if (list == NULL) return NULL;
  *list = (linkedList){ NULL, NULL };
  return list;
}

node* nodeInit(int data, node* previos, node* next) {
  node* newNode = malloc(sizeof(node));
  if (newNode == NULL) return NULL;
  *newNode = (node){ data, previos, next };
  return newNode;
}

node* listPush(linkedList* list, int data) {
  node* newNode = nodeInit(data, list->last, NULL);
  if (newNode == NULL) return NULL;
  if (list->first == NULL) list->first = newNode;
```

```
  else list->last->next = newNode;
  list->last = newNode;
  return newNode;
}

void listDestroy(linkedList* list) {
  node* curent = list->first;
  node* next = NULL;
  while (curent != NULL) {
    next = curent->next;
    free(curent);
    curent = next;
  }
  free(list);
}

int calculate(linkedList* list) {
  int result = 1;
  node* first = list->first;
  node* last = list->last;
  while (first->next != NULL) {
    result *= (first->data + first->next->data + 2 * last->data);
    first = first->next;
    last = last->previos;
  }
  return result;
}

int main(int argc, char const *argv[]) {
  linkedList* list = listInit();
  if (list == NULL) {
    printf("Error: There is not enough memory");
    return 0;
```

```
  }
  int length, data, success = 1;
  while (1) {
    printf("Enter the count of elements you are going to enter: ");
    scanf("%d", &length);
    if (length > 1) break;
    printf("The value must be greater than 1\n");
  }
  for (int i = 1; i <= length; i++) {
    printf("Enter the element #%d: ", i);
    scanf("%d", &data);
    node* item = listPush(list, data);
    if (item == NULL) {
      printf("Error: There is not enough memory");
      success = 0;
      break;
    }
  }
  if (success) {
    printf("Result: %d\n", calculate(list));
  }
  listDestroy(list);
  return 0;
}
```

Тестування програм

Формула:

$$(a_1 + a_2 + 2a_n)(a_2 + a_3 + 2a_{n-1}) \dots (a_{n-1} + a_n + 2a_2)$$

1. Реалізація на основі двохзв'язного циклічного списку:

   a) (1, 2, 3, 4, 5, 6):

$$(1 + 2 + 2 * 6)(2 + 3 + 2 * 5)(3 + 4 + 2 * 4)(4 + 5 + 2 * 3)(5 + 6 + 2 * 2)$$

$$= 15 * 15 * 15 * 15 * 15 = 15^5 = 759375$$

```
PS D:\programming\code\c> .\circularLinkedList.exe
Enter the count of elements you are going to enter: 6
Enter the element #1: 1
Enter the element #2: 2
Enter the element #3: 3
Enter the element #4: 4
Enter the element #5: 5
Enter the element #6: 6
Result: 759375
```

   b) (3, 2, 5, 4):

$$(3 + 2 + 2 * 4)(2 + 5 + 2 * 5)(5 + 4 + 2 * 2) = 13 * 17 * 13 = 2873$$

```
PS D:\programming\code\c> .\circularLinkedList.exe
Enter the count of elements you are going to enter: 4
Enter the element #1: 3
Enter the element #2: 2
Enter the element #3: 5
Enter the element #4: 4
Result: 2873
```

   c) (5, -1, -2, 3):

$$(5 + (-1) + 2 * 3)\big(-1 + (-2) + 2 * (-2)\big)\big(-2 + 3 + 2 * (-1)\big)$$

$$= 10 * (-7) * (-1) = 70$$

```
PS D:\programming\code\c> .\circularLinkedList.exe
Enter the count of elements you are going to enter: 4
Enter the element #1: 5
Enter the element #2: -1
Enter the element #3: -2
Enter the element #4: 3
Result: 70
```

2. Реалізація на основі двохзв’язного списку з заголовком:

a) (3, 6, 2, 4, 3, 1):

$$(3+6+2*1)(6+2\ +\ 2*\ 3)(2+4+2*4)(4+3+2*\ 2)(3+1+2*6)$$
$$= 11*14*14*11*16 = \ 15^5 = 379456$$

```
PS D:\programming\code\c> .\linkedList.exe
Enter the count of elements you are going to enter: 6
Enter the element #1: 3
Enter the element #2: 6
Enter the element #3: 2
Enter the element #4: 4
Enter the element #5: 3
Enter the element #6: 1
Result: 379456
```

b) (1, 2, 3, 4):

$$(1+2+2*4)(2+3+2*3)(3+4+2*2) = 11*11*11 = \ 11^3 = 1331$$

```
PS D:\programming\code\c> .\linkedList.exe
Enter the count of elements you are going to enter: 4
Enter the element #1: 1
Enter the element #2: 2
Enter the element #3: 3
Enter the element #4: 4
Result: 1331
```

c) (-2, -1, 0, 1):

$$(-2+(-1)+2*1)(-1+0+2*0)\big(0+1+2*(-1)\big)$$
$$= -1*(-1)*(-1) = -1$$

```
PS D:\programming\code\c> .\linkedList.exe
Enter the count of elements you are going to enter: 4
Enter the element #1: -2
Enter the element #2: -1
Enter the element #3: 0
Enter the element #4: 1
Result: -1
```

Результати збігаються, отже програми написані правильно.

Висновок

Протягом виконання лабораторної роботи я навчився розробляти зв'язані динамічні структури даних. Для вирішення своєї задачі я вибрав два типи списків: двохзв'язний циклічний список та двохзв'язного список з заголовком, адже мені потрібно мати доступ до сусідніх елементів в обидві сторони, та мати можливість звернутися до першого доданого елементу без циклу. Обидві ці структури задовільнять ці вимоги. Кожна реалізація має свої переваги та недоліки. Двохзв'язний циклічний список займає менше пам'яті, адже не потрібно зберігати окремо заголовок з двома вказівниками. З іншої сторони, у двохзв'язного списку з заголовком завжди є доступ до першого та останнього елементу, також за допомогою цього синтаксису можна зробити зручніший інтерфейс взаємодії зі списком. Проте, насправді, ці реалізації сильно не відрізняються ні по швидкодії ні по вимоги до пам'яті.

Не для всіх задач підходить статичний масив, адже не завжди ми можемо знати кількість даних на момент написання програми. Для вирішення цієї проблеми можна використовувати динамічні структури даних. Динамічний масив не завжди можна використовувати через низьку швидкість додавання та видалення нових елементів, також не завжди в пам'яті може знайтися безперервна вільна ділянка. Зв'язані структури даних вирішують обидві проблеми. Додавання та видалення елементів виконується за O(1), а самі дані не повинні зберігатися послідовно в пам'яті. Проте, такий спосіб зберігання даних має свої недоліки. Зв'язані структури даних займають більше пам'яті, адже окрім самих даних потрібно зберігати один або більше вказівників на наступні елементи. Також доступ до елементів списку відбувається за O(n).

Зв'язані структури даних слід використовувати коли значення часто додаються або видаляються зі списку, а доступ до даних всередині відбувається рідко.