Министерство науки и высшего образования Российской Федерации  
Пензенский государственный университет  
Кафедра вычислительная техника

**ОТЧËТ**  
по лабораторной работе №3  
по курсу «Логика и основы алгоритмизации в инженерных задачах»  
на тему «Динамические списки.»

Выполнили студенты группы 22ВВВ2:  
Хоссейни Нежад С. А. С. М.

Приняли:  
Акифьев И. В.  
Митрохин М. А.

Пенза 2023

**Название**

Динамические списки.

**Лабораторное задание**

Задание 1: Реализовать приоритетную очередь, путём добавления элемента в список в соответствии с приоритетом объекта (т.е. объект  с большим приоритетом становится перед объектом с меньшим приоритетом).

Задание 2: \* На основе приведенного кода реализуйте структуру данных Очередь.

Задание 3: \* На основе приведенного кода реализуйте структуру данных Стек.

**Ход работы**

Задание 1:

Написал две функции для добавления элемента с приоритетом.

* Для пользователя

void push()

{

struct node\* p = NULL;

p = create\_node();

push\_logic(p);

}

struct node\* create\_node()

{

struct node\* p = NULL;

int priority = -1;

char task[50];

int ext = 0;

if ((p = (node\*)malloc(sizeof(struct node))) == NULL)

{

printf("memory allocation error\n");

exit(1);

}

do

{

printf("Priority input: ");

scanf("%d", &priority);

if (priority == -1)

{

printf("no recording was made\n");

scanf("%\*[^\n]");

getchar();

continue;

}

getchar();

printf("Enter a task: ");

fgets(task, 50, stdin);

if (\*task == 0)

{

printf("no recording was made\n");

scanf("%\*[^\n]");

getchar();

continue;

}

ext = 1;

} while (ext == 0);

p->priority = priority;

strcpy(p->task, task);

p->next = NULL;

return p;

}

* Для программиста

void add\_element(int priority, char\* task)

{

struct node\* p = NULL;

p = create\_node\_list(priority, task);

push\_logic(p);

}

struct node\* create\_node\_list(int priority, char\* task)

{

struct node\* p = NULL;

if ((p = (node\*)malloc(sizeof(struct node))) == NULL)

{

printf("memory allocation error\n");

exit(1);

}

p->priority = priority;

strcpy(p->task, task);

p->next = NULL;

return p;

}

Общая логика работы:

void push\_logic(struct node\* p)

{

if (head == NULL && p != NULL)

{

head = p;

return;

}

else

{

struct node\* struc = head;

struct node\* prev;

if (p->priority > struc->priority)

{

prev = struc->next;

p->next = struc;

struc->next = prev;

head = p;

return;

}

while (struc)

{

prev = struc;

struc = struc->next;

if (struc)

{

if (p->priority < prev->priority && p->priority > struc->priority)

{

prev->next = p;

p->next = struc;

break;

}

}

else

{

struc = prev;

if (p->priority > struc->priority)

{

p->next = struc;

struc->next = NULL;

}

else

{

struc->next = p;

p->next = NULL;

}

break;

}

}

}

}

* Взятие элемента:

Для пользователя

void pop()

{

if (head == NULL)

{

printf("the list is empty\n");

return;

}

struct node\* struc = head;

head = struc->next;

printf("Info: %s", struc->inf);

free(struc);

printf("Press any key to exit");

}

Для программиста

char\* take\_element()

{

if (head == NULL)

{

printf("the list is empty\n");

return NULL;

}

char\* data = (char\*)malloc(50 \* sizeof(char));

struct node\* struc = head;

head = struc->next;

strcpy(data, struc->task);

free(struc);

return data;

}

Задание 2:

На основе приведенного кода реализовал структуру данных Очередь.

* Для пользователя

void push()

{

struct node\* p = NULL;

p = create\_node();

if (head == NULL && p != NULL)

{

head = p;

last = p;

}

else if (head != NULL && p != NULL)

{

last->next = p;

last = p;

}

return;

}

struct node\* create\_node()

{

struct node\* p = NULL;

char inf[256];

int ext = 0;

if ((p = (node\*)malloc(sizeof(struct node))) == NULL)

{

printf("memory allocation error\n");

exit(1);

}

do

{

printf("Enter a task: ");

fgets(inf, 256, stdin);

if (\*inf == 0)

{

printf("no recording was made\n");

scanf("%\*[^\n]");

getchar();

continue;

}

ext = 1;

} while (ext == 0);

strcpy(p->inf, inf);

p->next = NULL;

return p;

}

* Для программиста

void add\_element(char\* inf)

{

struct node\* p = NULL;

p = create\_node\_list(inf);

if (head == NULL && p != NULL)

{

head = p;

last = p;

}

else if (head != NULL && p != NULL)

{

last->next = p;

last = p;

}

return;

}

struct node\* create\_node\_list(char\* inf)

{

struct node\* p = NULL;

int ext = 0;

if ((p = (node\*)malloc(sizeof(struct node))) == NULL)

{

printf("memory allocation error\n");

exit(1);

}

strcpy(p->inf, inf);

p->next = NULL;

return p;

}

* Взятие элемента:

Для пользователя:

void pop()

{

if (head == NULL)

{

printf("the list is empty\n");

return;

}

struct node\* struc = head;

head = struc->next;

printf("Info: %s", struc->inf);

free(struc);

printf("Press any key to exit");

}

Для программиста:

char\* take\_element()

{

if (head == NULL)

{

printf("the list is empty\n");

return NULL;

}

char\* data = (char\*)malloc(256 \* sizeof(char));

struct node\* struc = head;

head = struc->next;

strcpy(data, struc->inf);

free(struc);

return data;

}

Задание 3:

На основе приведенного кода реализовал структуру данных Стек.

* Для пользователя

void push()

{

struct node\* p = NULL;

p = create\_node();

if (head == NULL && p != NULL)

{

head = p;

}

else if (head != NULL && p != NULL)

{

p->next = head;

head = p;

}

return;

}

struct node\* create\_node()

{

struct node\* p = NULL;

char inf[256];

int ext = 0;

if ((p = (node\*)malloc(sizeof(struct node))) == NULL)

{

printf("memory allocation error\n");

exit(1);

}

do

{

printf("Enter a task: ");

fgets(inf, 256, stdin);

if (\*inf == 0)

{

printf("no recording was made\n");

scanf("%\*[^\n]");

getchar();

continue;

}

ext = 1;

} while (ext == 0);

strcpy(p->inf, inf);

p->next = NULL;

return p;

}

* Для программиста

void add\_element(char\* inf)

{

struct node\* p = NULL;

p = create\_node\_list(inf);

if (head == NULL && p != NULL)

{

head = p;

}

else if (head != NULL && p != NULL)

{

p->next = head;

head = p;

}

return;

}

struct node\* create\_node\_list(char\* inf)

{

struct node\* p = NULL;

int ext = 0;

if ((p = (node\*)malloc(sizeof(struct node))) == NULL)

{

printf("memory allocation error\n");

exit(1);

}

strcpy(p->inf, inf);

p->next = NULL;

return p;

}

* Взятие элемента:

Для пользователя:

void pop()

{

if (head == NULL)

{

printf("the list is empty\n");

return;

}

struct node\* struc = head;

head = struc->next;

printf("Info: %s", struc->inf);

free(struc);

printf("Press any key to exit");

}

Для программиста:

void take\_element()

{

if (head == NULL)

{

printf("the list is empty\n");

return;

}

char\* data = (char\*)malloc(256 \* sizeof(char));

struct node\* struc = head;

head = struc->next;

free(struc);

}

**Листинг**

* Задание 1

#define \_CRT\_SECURE\_NO\_WARNINGS

#include <stdio.h>

#include <stdlib.h>

#include <time.h>

#include <string.h>

void push();

void review();

void pop();

void del(int priority);

struct node\* create\_node();

struct node\* create\_node\_list(int priority, char\* task);

void add\_element(int priority, char\* task);

void push\_logic(struct node\* p);

char\* take\_element();

struct node\* head = NULL;

int main()

{

char ch;

int ext = 0, extt = 0, priority = -1;

do

{

system("cls");

printf("1. Add element\n2. Take element\n3. View list\n4. Delete element\n");

printf("Select item: ");

scanf("%c", &ch);

system("cls");

if (ch != '\n')

{

scanf("%\*[^\n]");

getchar();

}

switch (ch)

{

case '1':

push();

break;

case '2':

pop();

printf("Press any key to exit");

getchar();

break;

case '3':

review();

printf("Press any key to exit");

getchar();

break;

case '4':

do

{

printf("Enter the priority: ");

scanf("%d", &priority);

if (priority == -1)

{

printf("no recording was made\n");

scanf("%\*[^\n]");

getchar();

continue;

}

getchar();

extt = 1;

} while (extt == 0);

del(priority);

printf("Press any key to exit");

getchar();

break;

default:

break;

}

} while (ext == 0);

return 0;

}

struct node

{

int priority;

char task[50];

struct node\* next;

};

struct node\* create\_node()

{

struct node\* p = NULL;

int priority = -1;

char task[50];

int ext = 0;

if ((p = (node\*)malloc(sizeof(struct node))) == NULL)

{

printf("memory allocation error\n");

exit(1);

}

do

{

printf("Priority input: ");

scanf("%d", &priority);

if (priority == -1)

{

printf("no recording was made\n");

scanf("%\*[^\n]");

getchar();

continue;

}

getchar();

printf("Enter a task: ");

fgets(task, 50, stdin);

if (\*task == 0)

{

printf("no recording was made\n");

scanf("%\*[^\n]");

getchar();

continue;

}

ext = 1;

} while (ext == 0);

p->priority = priority;

strcpy(p->task, task);

p->next = NULL;

return p;

}

void review()

{

struct node\* struc = head;

if (head == NULL)

{

printf("the list is empty\n");

}

while (struc)

{

printf("Priority: %d, \n", struc->priority);

printf("Task: %s", struc->task);

struc = struc->next;

}

}

void del(int priority)

{

struct node\* struc = head;

struct node\* prev = struc;

int flag = 0;

if (head == NULL)

{

printf("the list is empty\n");

return;

}

if (struc->priority == priority)

{

flag = 1;

head = struc->next;

free(struc);

struc = head;

}

else

{

prev = struc;

struc = struc->next;

}

while (struc)

{

if (struc->priority == priority)

{

flag = 1;

if (struc->next)

{

prev->next = struc->next;

free(struc);

struc = prev->next;

}

else

{

prev->next = NULL;

free(struc);

return;

}

}

else

{

prev = struc;

struc = struc->next;

}

}

if (flag == 0)

{

printf("element not found\n");

return;

}

}

void push()

{

struct node\* p = NULL;

p = create\_node();

push\_logic(p);

}

void pop()

{

if (head == NULL)

{

printf("the list is empty\n");

return;

}

struct node\* struc = head;

head = struc->next;

printf("Priority: %d, \n", struc->priority);

printf("Task: %s", struc->task);

free(struc);

}

char\* take\_element()

{

if (head == NULL)

{

printf("the list is empty\n");

return NULL;

}

char\* data = (char\*)malloc(50 \* sizeof(char));

struct node\* struc = head;

head = struc->next;

strcpy(data, struc->task);

free(struc);

return data;

}

struct node\* create\_node\_list(int priority, char\* task)

{

struct node\* p = NULL;

if ((p = (node\*)malloc(sizeof(struct node))) == NULL)

{

printf("memory allocation error\n");

exit(1);

}

p->priority = priority;

strcpy(p->task, task);

p->next = NULL;

return p;

}

void add\_element(int priority, char\* task)

{

struct node\* p = NULL;

p = create\_node\_list(priority, task);

push\_logic(p);

}

void push\_logic(struct node\* p)

{

if (head == NULL && p != NULL)

{

head = p;

return;

}

else

{

struct node\* struc = head;

struct node\* prev;

if (p->priority > struc->priority)

{

prev = struc->next;

p->next = struc;

struc->next = prev;

head = p;

return;

}

while (struc)

{

prev = struc;

struc = struc->next;

if (struc)

{

if (p->priority < prev->priority && p->priority > struc->priority)

{

prev->next = p;

p->next = struc;

break;

}

}

else

{

struc = prev;

if (p->priority > struc->priority)

{

p->next = struc;

struc->next = NULL;

}

else

{

struc->next = p;

p->next = NULL;

}

break;

}

}

}

}

* Задание 2

#define \_CRT\_SECURE\_NO\_WARNINGS

#include <stdio.h>

#include <stdlib.h>

#include <time.h>

#include <string.h>

void push();

void review();

void del(char\* name);

struct node\* create\_node();

void find(char\* name);

void pop();

struct node\* create\_node\_list(char\* inf);

char\* take\_element();

void add\_element(char\* inf);

struct node\* head = NULL, \* last = NULL;

int main()

{

char ch;

int exit = 0;

char inf[256];

do

{

system("cls");

printf("1. Add element\n2. Take element\n3. View list\n4. Find element\n5. Delete element\n");

printf("Select item: ");

scanf("%c", &ch);

system("cls");

if (ch != '\n')

{

scanf("%\*[^\n]");

getchar();

}

switch (ch)

{

case '1':

push();

break;

case '2':

pop();

getchar();

break;

case '3':

review();

getchar();

break;

case '4':

printf("Enter the element name: ");

fgets(inf, 256, stdin);

find(inf);

printf("Press any key to exit");

getchar();

break;

case '5':

printf("Enter the element name: ");

fgets(inf, 256, stdin);

del(inf);

printf("Press any key to exit");

getchar();

break;

default:

break;

}

} while (exit == 0);

return 0;

}

struct node

{

char inf[256];

struct node\* next;

};

struct node\* create\_node()

{

struct node\* p = NULL;

char inf[256];

int ext = 0;

if ((p = (node\*)malloc(sizeof(struct node))) == NULL)

{

printf("memory allocation error\n");

exit(1);

}

do

{

printf("Enter a task: ");

fgets(inf, 256, stdin);

if (\*inf == 0)

{

printf("no recording was made\n");

scanf("%\*[^\n]");

getchar();

continue;

}

ext = 1;

} while (ext == 0);

strcpy(p->inf, inf);

p->next = NULL;

return p;

}

void review()

{

struct node\* struc = head;

if (head == NULL)

{

printf("the list is empty\n");

}

while (struc)

{

printf("Task: %s", struc->inf);

struc = struc->next;

}

printf("Press any key to exit");

}

void push()

{

struct node\* p = NULL;

p = create\_node();

if (head == NULL && p != NULL)

{

head = p;

last = p;

}

else if (head != NULL && p != NULL)

{

last->next = p;

last = p;

}

return;

}

void find(char\* name)

{

struct node\* struc = head;

if (head == NULL)

{

printf("the list is empty\n");

}

while (struc)

{

if (strcmp(name, struc->inf) == 0)

{

printf("Element found");

printf("\nInfo: %s", struc->inf);

return;

}

struc = struc->next;

}

printf("Element not found\n");

return;

}

void del(char\* name)

{

struct node\* struc = head;

struct node\* prev = struc;

int flag = 0;

if (head == NULL)

{

printf("the list is empty\n");

return;

}

if (strcmp(name, struc->inf) == 0)

{

flag = 1;

printf("Successful element deletion %s", struc->inf);

head = struc->next;

free(struc);

struc = head;

}

else

{

prev = struc;

struc = struc->next;

}

while (struc)

{

if (strcmp(name, struc->inf) == 0)

{

flag = 1;

if (struc->next)

{

printf("Successful element deletion %s", struc->inf);

prev->next = struc->next;

free(struc);

struc = prev->next;

}

else

{

printf("Successful element deletion %s", struc->inf);

prev->next = NULL;

free(struc);

return;

}

}

else

{

prev = struc;

struc = struc->next;

}

}

if (flag == 0)

{

printf("Element not found\n");

return;

}

}

void pop()

{

if (head == NULL)

{

printf("the list is empty\n");

return;

}

struct node\* struc = head;

head = struc->next;

printf("Info: %s", struc->inf);

free(struc);

printf("Press any key to exit");

}

void add\_element(char\* inf)

{

struct node\* p = NULL;

p = create\_node\_list(inf);

if (head == NULL && p != NULL)

{

head = p;

last = p;

}

else if (head != NULL && p != NULL)

{

last->next = p;

last = p;

}

return;

}

char\* take\_element()

{

if (head == NULL)

{

printf("the list is empty\n");

return NULL;

}

char\* data = (char\*)malloc(256 \* sizeof(char));

struct node\* struc = head;

head = struc->next;

strcpy(data, struc->inf);

free(struc);

return data;

}

struct node\* create\_node\_list(char\* inf)

{

struct node\* p = NULL;

int ext = 0;

if ((p = (node\*)malloc(sizeof(struct node))) == NULL)

{

printf("memory allocation error\n");

exit(1);

}

strcpy(p->inf, inf);

p->next = NULL;

return p;

}

* Задание 3

#define \_CRT\_SECURE\_NO\_WARNINGS

#include <stdio.h>

#include <stdlib.h>

#include <time.h>

#include <string.h>

void push();

void review();

void del(char\* name);

struct node\* create\_node();

void find(char\* name);

void pop();

struct node\* create\_node\_list(char\* inf);

void take\_element();

void add\_element(char\* inf);

struct node\* head = NULL;

int main()

{

char ch;

int ext = 0;

char inf[256];

do

{

system("cls");

printf("1. Add element\n2. Take element\n3. View list\n4. Find element\n5. Delete element\n");

printf("Select item: ");

scanf("%c", &ch);

system("cls");

if (ch != '\n')

{

scanf("%\*[^\n]");

getchar();

}

switch (ch)

{

case '1':

push();

break;

case '2':

pop();

getchar();

break;

case '3':

review();

getchar();

break;

case '4':

printf("Enter the element name: ");

fgets(inf, 256, stdin);

find(inf);

printf("Press any key to exit");

getchar();

break;

case '5':

printf("Enter the element name: ");

fgets(inf, 256, stdin);

del(inf);

printf("Press any key to exit");

getchar();

break;

default:

break;

}

} while (ext == 0);

return 0;

}

struct node

{

char inf[256];

struct node\* next;

};

struct node\* create\_node()

{

struct node\* p = NULL;

char inf[256];

int ext = 0;

if ((p = (node\*)malloc(sizeof(struct node))) == NULL)

{

printf("memory allocation error\n");

exit(1);

}

do

{

printf("Enter a task: ");

fgets(inf, 256, stdin);

if (\*inf == 0)

{

printf("no recording was made\n");

scanf("%\*[^\n]");

getchar();

continue;

}

ext = 1;

} while (ext == 0);

strcpy(p->inf, inf);

p->next = NULL;

return p;

}

void review()

{

struct node\* struc = head;

if (head == NULL)

{

printf("the list is empty\n");

}

while (struc)

{

printf("Task: %s", struc->inf);

struc = struc->next;

}

printf("Press any key to exit");

}

void push()

{

struct node\* p = NULL;

p = create\_node();

if (head == NULL && p != NULL)

{

head = p;

}

else if (head != NULL && p != NULL)

{

p->next = head;

head = p;

}

return;

}

void find(char\* name)

{

struct node\* struc = head;

if (head == NULL)

{

printf("the list is empty\n");

}

while (struc)

{

if (strcmp(name, struc->inf) == 0)

{

printf("Element found");

printf("\nInfo: %s", struc->inf);

return;

}

struc = struc->next;

}

printf("Element not found\n");

return;

}

void del(char\* name)

{

struct node\* struc = head;

struct node\* prev = struc;

int flag = 0;

if (head == NULL)

{

printf("the list is empty\n");

return;

}

if (strcmp(name, struc->inf) == 0)

{

flag = 1;

printf("Successful element deletion %s", struc->inf);

head = struc->next;

free(struc);

struc = head;

}

else

{

prev = struc;

struc = struc->next;

}

while (struc)

{

if (strcmp(name, struc->inf) == 0)

{

flag = 1;

if (struc->next)

{

printf("Successful element deletion %s", struc->inf);

prev->next = struc->next;

free(struc);

struc = prev->next;

}

else

{

printf("Successful element deletion %s", struc->inf);

prev->next = NULL;

free(struc);

return;

}

}

else

{

prev = struc;

struc = struc->next;

}

}

if (flag == 0)

{

printf("Element not found\n");

return;

}

}

void pop()

{

if (head == NULL)

{

printf("the list is empty\n");

return;

}

struct node\* struc = head;

head = struc->next;

printf("Info: %s", struc->inf);

free(struc);

printf("Press any key to exit");

}

void add\_element(char\* inf)

{

struct node\* p = NULL;

p = create\_node\_list(inf);

if (head == NULL && p != NULL)

{

head = p;

}

else if (head != NULL && p != NULL)

{

p->next = head;

head = p;

}

return;

}

void take\_element()

{

if (head == NULL)

{

printf("the list is empty\n");

return;

}

char\* data = (char\*)malloc(256 \* sizeof(char));

struct node\* struc = head;

head = struc->next;

free(struc);

}

struct node\* create\_node\_list(char\* inf)

{

struct node\* p = NULL;

int ext = 0;

if ((p = (node\*)malloc(sizeof(struct node))) == NULL)

{

printf("memory allocation error\n");

exit(1);

}

strcpy(p->inf, inf);

p->next = NULL;

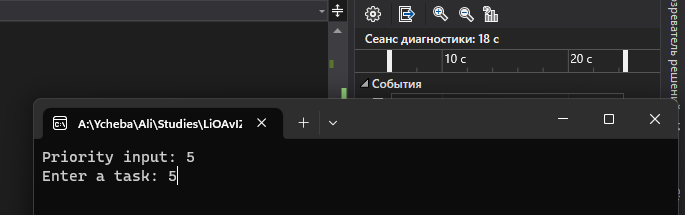
return p;

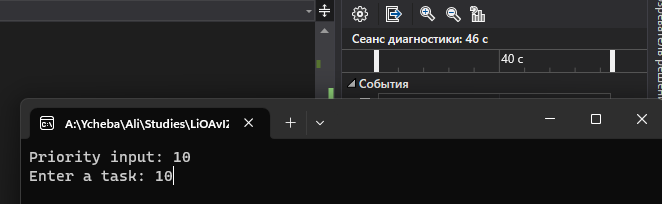
}

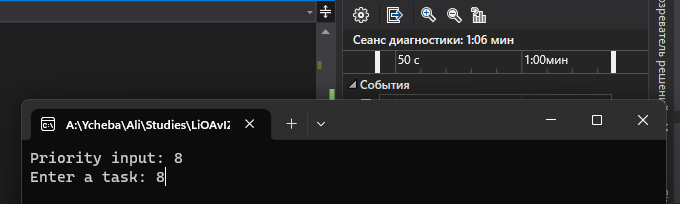
**Результат работы программы**

* Задание 1

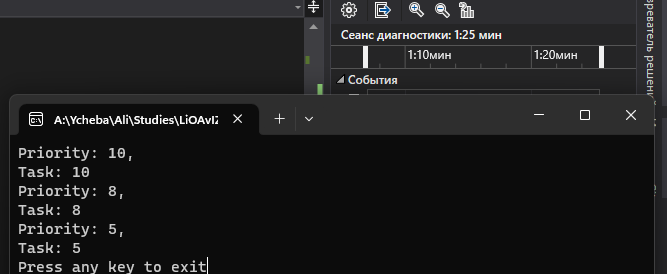
Добавляем элементы

****

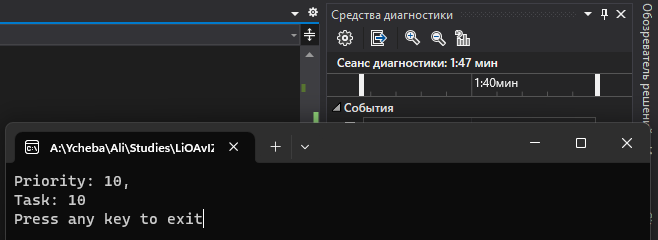
****

****

Смотрим содержимое

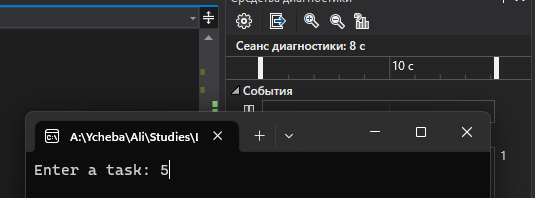
****

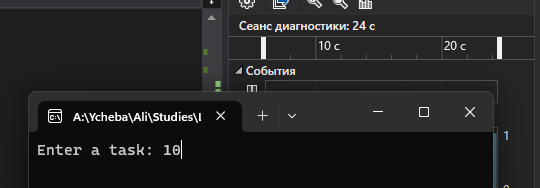
Берем элемент

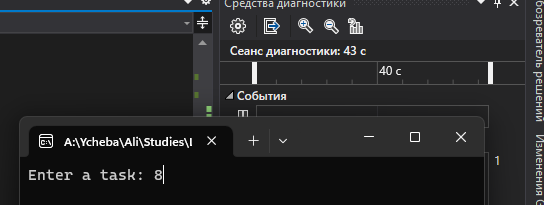
****

* Задание 2

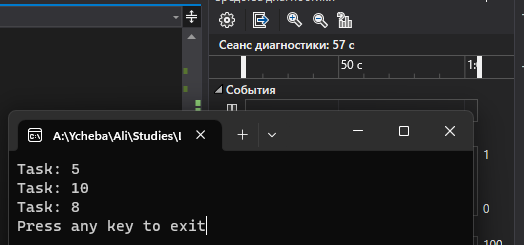
Добавляем элементы



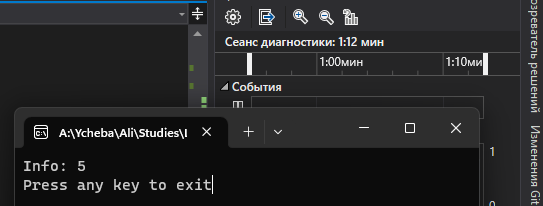




Смотрим содержимое

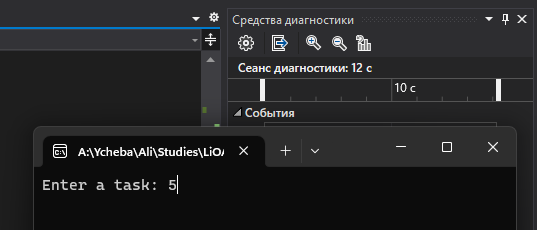


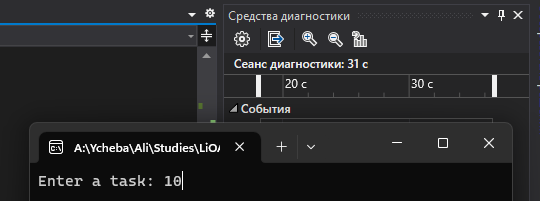
Берем элемент

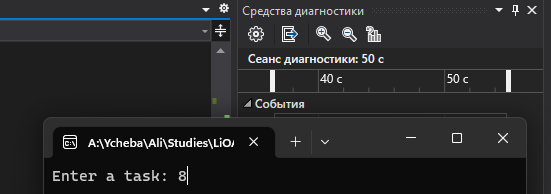


* Задание 3

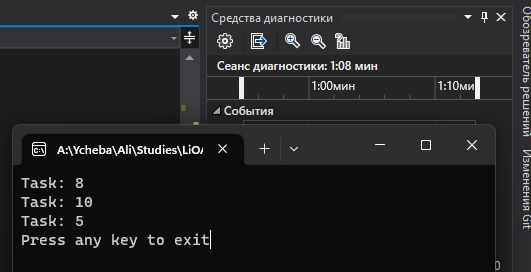
Добавляем элементы



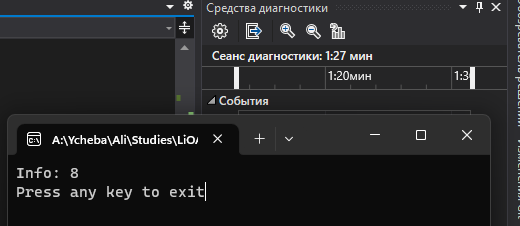




Смотрим содержимое



Берем элемент



**Вывод**

В результате выполнения лабораторной работы я изучил динамические списки.