# ЦЕЛЬ РАБОТЫ

Изучение принципов организации и программирование основных операций, выполняемых с линейными списковыми структурами в языке С.

# ПОСТАНОВКА ЗАДАЧИ

1. Согласно варианту №7 требуется разработать программу формирования однонаправленного списка и выполнения операции перестановки *k* -того и (*k +1)*-ого элементов списка.
2. В программе предусмотреть вывод элементов исходного списка и результирующего списка.

# ОПИСАНИЕ ИСПОЛЬЗУЕМЫХ СТРУКТУР ДАННЫХ

Для представления списка была использована однонаправленная очередь, представляющая собой линейную структуру, в которой все включения производятся в конец структуры, а все исключения в начале. В программной реализации на языке C, структура имеет вид:

struct list{  
 int info;  
 struct list \*next;  
}

где поле info – информационное поле целочисленного типа, а поле next – указатель на структуру типа list.

# ОПИСАНИЕ АЛГОРИТМА РЕШЕНИЯ ЗАДАЧИ



Рисунок 1 – Структурная схема алгоритма обмена соседних элементов списка местами

# СПЕЦИФИКАЦИИ ПОДПРОГРАММ

Во всех функциях данной программы, переменная *beg* выступает в роли указателя на начало списка, а переменная *end* указателя на конец списка.

Функции, которые предусматривают добавление, удаление элементов и изменение их порядка следования, требуют передачи параметров по ссылке.

Функция swapNodes принимает в качестве аргументов 3 переменные, *beg* и *end* – указатели на начало и конец списка соответственно, а swapedNode номер *k-ого* элемента.

# ТЕКСТ ПРОГРАММЫ

#include <stdio.h>  
#include <stdlib.h>  
#include <conio.h>  
  
struct list{  
 int info;  
 struct list \*next;  
} \*beg = NULL, \*end = NULL;  
  
static size\_t recCounter = 0;

struct list \*addNodeToEnd(struct list \*\*beg, struct list \*\*end, int nodeInfo){  
 struct list \*temp = calloc(1,sizeof(struct list));  
 //temp->info = nodeInfo;  
 temp->info = recCounter+1;  
 temp->next = NULL;  
  
 if (\*beg == NULL){  
 \*beg = temp;  
 \*end = \*beg;  
 } else {  
 (\*end)->next = temp;  
 \*end = temp;  
 }

++recCounter;  
 return \*beg;  
}  
  
struct list \*deleteNodeFromBeg(struct list \*\*beg){  
 if (beg != NULL){  
 struct list \*temp = \*beg;  
 \*beg = temp->next;  
 free(temp);  
 --recCounter;  
 }  
 return \*beg;  
}  
  
int viewlist(struct list \*beg){  
 if (beg == NULL){  
 return 0 ;  
 }  
 printf("\n# | Info \n");  
 struct list \*temp = beg;  
 size\_t counter = 1;  
 while (temp != NULL){  
 printf("%d. %d \n", counter, temp->info);  
 temp = temp->next;  
 ++counter;  
 }  
 return 1;  
}  
  
struct list \*swapNodes(struct list \*\*beg, struct list \*\*end, size\_t swapedNode){  
 if ((\*beg == NULL) || (!swapedNode) || (swapedNode >= recCounter)){  
 return \*beg;  
 }  
 struct list \*temp1 = \*beg, \*temp2;  
 size\_t i = 1;  
  
 if (swapedNode == 1){  
 temp2 = temp1->next;  
 temp1->next = temp2->next;  
 temp2->next = temp1;  
 \*beg = temp2;  
 } else {  
 while(i < (swapedNode - 1)){  
 temp1 = temp1->next;  
 ++i;  
 }  
 temp2 = temp1->next;  
 temp1->next = temp2->next;  
 temp2->next = temp2->next->next;  
 temp1->next->next = temp2;  
 if(temp2->next == NULL){  
 \*end = temp2;  
 }  
 }  
 return \*beg;  
}  
  
int main() {  
 while (1){  
 system("cls");  
 printf("1-Add node to end\n");  
 printf("2-Delete first node\n");  
 printf("3-Move k to k+1 node\n");  
 printf("4-View list\n");  
 printf("0-Exit\n");  
 char key;  
  
 key = getch();  
 switch(key){  
 case '1':{  
 int tempInfo = 0;   
 if (addNodeToEnd(&beg,&end,tempInfo) != NULL){  
 printf("Node added to end of list\n");  
 }else{  
 printf("ERROR\n");  
 }  
 if (beg != NULL){  
 viewlist(beg);  
 }  
 break;  
 }  
 case '2':{  
 if (beg != NULL) {  
 deleteNodeFromBeg(&beg);  
 printf("First node deleted\n");  
 } else{  
 printf("ERROR\n");  
 }  
 if (beg != NULL){  
 viewlist(beg);  
 } else {  
 printf("EMPTY\n");  
 } break; }  
 case '3':{  
 if(beg != NULL){  
 size\_t nodeNum = 0;  
 printf("Enter k-number of node: ");  
 scanf("%d",&nodeNum);  
 if ((nodeNum < recCounter) && (nodeNum > 0)){  
 swapNodes(&beg, &end, nodeNum);  
 viewlist(beg);  
 } else {  
 printf("ERROR\n");  
 }  
 } break; }  
 case '4':{  
 if (beg != NULL){  
 viewlist(beg);  
 } else {  
 printf("ERROR\n");  
 } break; }  
 case '0':{  
 return 0;  
 } } getch(); }  
}

# ТЕСТИРОВАНИЕ ПРОГРАММЫ

Для полной проверки работоспособности программы было составлено 5 тестов:

1. Удаление первого элемента (рис.3) и последующих элементов списка (рис.4) не вызывает аварийного завершения программы, а так же не оставляет «висячих» ссылок:

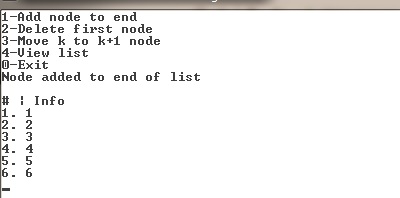


Рисунок 2 – Исходный вид списка

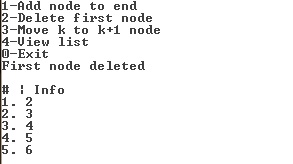


Рисунок 3 – Результат удаления первого элемента списка

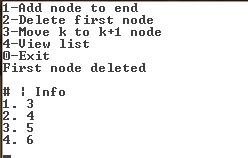


Рисунок 4 – Результат удаления элемента из середины списка

1. Удаление последнего элемента списка, так же не приводит к нарушению работоспособности программы (рис.5):

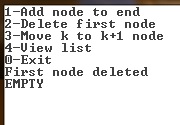


Рисунок 5 – Результат удаления последнего элемента списка

1. Для исходного списка (рис.2) была произведена перестановка предпоследнего и последнего элемента, что не привело к ошибке(рис.6):

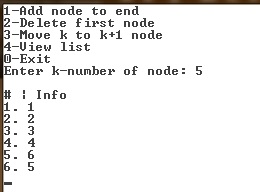


Рисунок 6 – Результат удаления последнего элемента списка

1. Для списка, состоящего из двух элементов (рис.7) была выполнена перестановка этих элементов (рис.8):

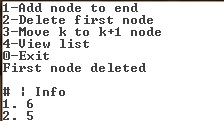


Рисунок 7 – Исходный вид списка из 2 элементов

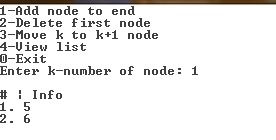


Рисунок 8 – Результат перестановки элементов исходного списка

ВЫВОД

При выполнении лабораторной работы была разработана, а так же протестирована программа написанная на языке C, позволяющая сформировать однонаправленный список. Программа позволяет добавлять, а так же удалять элементы, согласно специфике работы однонаправленных очередей. Результат работы подпрограммы выводится на экран. Так же были предусмотрены обработки ошибок, таких как: просмотр и удаление элементов из пустого списка, а так перестановка элементов с несуществующими индексами.

По результатам тестирования, можно сделать вывод, что программа работоспособна и поставленная задача выполнена.