Задание 1

Главным вопросом, волнующим руководство «Предприятия», в последнее время стал вопрос использования статических структур данных – массивов. В соответствии с докладом председателя комитета по структурам данных, массивы обладают следующими недостатками: (1) невозможность удаления или добавления элемента без сдвига других; (2) для динамического массива – дополнительные накладные расходы на поддержку динамических свойств; (3) угроза выхода за границы массива и повреждения данных. Также в докладе прозвучала информация о том, что при решении некоторых задач могут использоваться связные списки, для которых нет недостатков, имеющихся у массивов. Несмотря на то, что связные списки имеют свои недостатки, руководство корпорации решило изучить порядок работы со связными списками.

Руководство «Предприятия» поставило задачу написать программу, реализующую операции со списками.

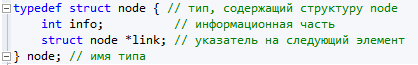
Отдел постановки задач департамента разработки программного обеспечения сформулировал для вас как для исполнителя следующие требования в техническом задании.

1) Программа должна иметь имя 20ХХ\_SAS03\_Фамилия, где фамилия – фамилия автора программы на латинице.

2) Программа должна использовать функции, разработанные непосредственно исполнителем.

3) При создании программы руководствоваться следующими положениями:

3.1.) Каждый элемент списка описывается структурой, содержащей информационную часть и указатель на следующий элемент списка. Структура должна быть объявлена и описана вне всяких функций (в самом верху сразу под разделом с подключением модулей):



Оператор typedef позволяет создать пользовательский тип. Как видно, имя структуры и имя типа совпадают. Это не ошибка. Это допускается в языке С++. Благодаря созданию своего типа появляется возможность объявлять переменные так:

node \*first;

Это гораздо удобнее, чем такое объявление без использования типа:

struct node \*first;

3.2.) Программа должна реализовывать следующие операции со списком:

- добавление в начало списка (функция addFirstNode);

- добавление в конец списка (функция addLastNode);

- добавление после заданного элемента (функция addAfterNode);

- удаление первого элемента списка (функция deleteFirstNode);

- удаление последнего элемента списка (функция deleteLastNode);

- удаление заданного элемента списка (deleteNode);

- просмотр списка (функция printList);

3.3.) Порядок разработки программы должен быть следующим:

Данная программа имеет такую же структуру, как и предыдущие, с той лишь разницей, что все действия выполняются со списком.

Рассмотрим главную функцию. Комментарии к функции даны ниже. Разберите каждый оператор функции, чтобы иметь возможность ответить, что происходит в каждой строке. Кто просто скопирует это и вставит, ничего дальше не сможет сделать.

int main()

{

setlocale(LC\_ALL, ".1251");

char ch;

int m, n;

node \*first; // указатель на первый элемент списка

do {

showMenu(); // вывод меню

ch = getch();

printf("%c\n", ch);

switch (ch) {

case '0':

break;

case '1':

printf("Введите целое число для добавления в начало списка: ");

scanf("%d", &m);

if (addFirstNode(&first, m) != NULL) { // вызов функции добавления элемента в начало

printf("Операция выполнена успешно. Для продолжения нажмите любую клавишу.\n");

} else {

printf("Не удалось добавить новый элемент.");

}

getch();

break;

case '2':

printf("Введите целое число для добавления в конец списка: ");

scanf("%d", &m);

if (addLastNode(&first, m) != NULL) { // вызов функции добавления в конец

printf("Операция выполнена успешно. Для продолжения нажмите любую клавишу.\n");

} else {

printf("Не удалось добавить новый элемент. Программа будет закрыта.");

}

getch();

break;

case '3':

if (first != NULL) {

printf("Удален элемент со значением %d. Для продолжения нажмите любую клавишу.\n", deleteFirstNode(&first)); // удаление первого элемента списка

} else {

printf("Список не содержит элементов. Для продолжения нажмите любую клавишу.\n");

}

getch();

break;

case '4':

if (first != NULL) {

printf("Удален элемент со значением %d. Для продолжения нажмите любую клавишу.\n", deleteLastNode(&first)); // удаление последнего элемента списка

} else {

printf("Список не содержит элементов. Для продолжения нажмите любую клавишу.\n");

}

getch();

break;

case '5':

if (first != NULL) {

printList(first); // вывод списка на экран

} else {

printf("Список пуст. Для продолжения нажмите любую клавишу.\n");

}

getch();

break;

default:

printf("Такой операции не существует. Выберите другую операцию. Для продолжения нажмите любую клавишу.");

getch();

break;

}

} while (ch != '0');

return 0;

}

В первую очередь обратим внимание на то, как объявлен первый элемент списка:

node \*first;

Он объявлен как указатель. Иначе никак: у нас должна быть возможность проверять как значения самих элементов структуры info и link (это мы можем легко сделать, если переменная объявлена не как указатель: first.info и first.link). Но проверить сам элемент у нас нет никакой возможности. Если first объявлен как значение, можно обратиться к его элементам, а можно его куда-нибудь присвоить. Либо что-нибудь присвоить ему. Список же подразумевает, что может сложиться ситуация, когда список пуст. Тогда указатель на первый элемент не должен содержать ничего. При объявлении node first переменная first всегда содержит что-нибудь. Поэтому нет возможности проверить список на пустоту.

При использовании же указателя на структуру node \*first мы можем как обратиться к элементам этой структуры (first->info и first->link), и, так как указатель может указывать никуда, можно оценить саму переменную first (first == NULL и first != NULL).

Использование указателя повлечёт за собой некоторые трудности, но предоставит более гибкое управление списком.

Возьмите и скопируйте себе функцию вывода меню на экран, чтобы не печатать лишнего:

void showMenu() {

system("cls");

printf("1. Добавить в начало списка\n");

printf("2. Добавить в конец списка\n");

printf("3. Удалить элемент в начале списка\n");

printf("4. Удалить элемент в конце списка\n");

printf("5. Просмотреть содержимое очереди\n");

printf("0. Выход\n\n");

printf("Выберите операцию: ");

return;

}

Первая функция, которую нужно реализовать – добавление в начало списка. Она вызывается так:

addFirstNode(&first, m)

В функцию передается адрес указателя на first, т.к. внутри функции значение первого элемента списка может изменяться.

Разберемся, что будет передаваться в функцию. Вот память:

**\*first**

**адрес A**

адрес K

**адрес K**

**адрес L**

info

link

С адреса K начинаются элементы структуры. По адресу A расположен указатель first и он содержит адрес K – указывает на первый элемент списка. Таким образом, передавая в функцию &first, мы передаем в нее адрес А.

Что же необходимо принимать в функции addFirstNode? На этот вопрос ответить не так-то и легко.

Режим работы с указателями при работе со структурами является нормальным и правильным. Передавая в функцию указатель на структуру, и, принимая указатель структуры, мы, по сути, осуществляем передачу по значению. Так мы сможем изменять значения элементов структуры, но не сам элемент списка.

Для того чтобы иметь возможность изменить сам элемент списка, нужно, чтобы что-то указывало на этот элемент. То есть нужен как обычно при работе с обычными переменными указатель. А раз элемент структуры first сам по себе является указателем, то нужен указатель на указатель.

node\* addFirst(node \*\*first, int m)

Так объявлена функция addFirst. При такой передаче параметров внутри функции мы сможем как обратиться к элементам first и поменять их:

(\*first)->info = 6;

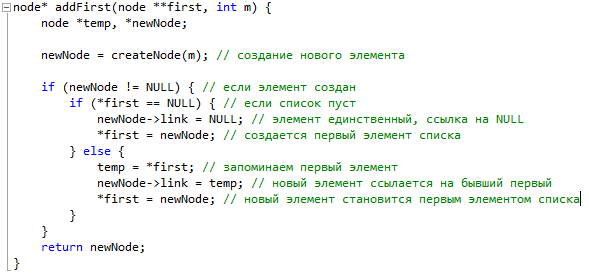
так и поменять сам элемент списка first:

node \*newNode ...

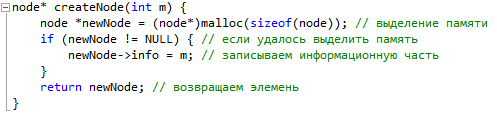
\*first = newNode;

При другом способе передачи функция работала бы с копией элемента списка, а не с ним самим.

Порассуждаем о добавлении элемента в список. Для того чтобы добавить элемент в список, его нужно создать (выделить под него память и заполнить её значениями). Далее возможны две ситуации: либо список пустой, тогда созданный элемент станет первым и единственным элементом списка, либо в списке уже есть элементы, тогда созданный элемент должен добавиться перед уже существующими.



В данной функции вызывается функция createNode, которая создает и возвращает новый элемент и записывает в его информационную часть m.

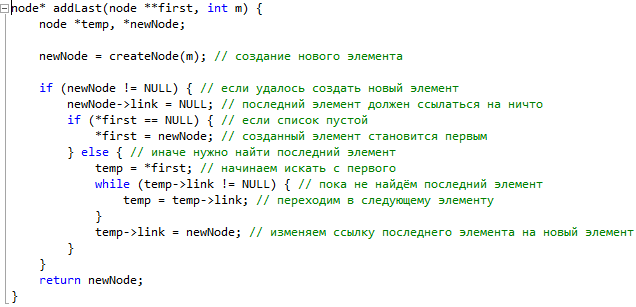


Используемая функция malloc выделяет память заданного объема под значения определенного типа. Память выделяется так:

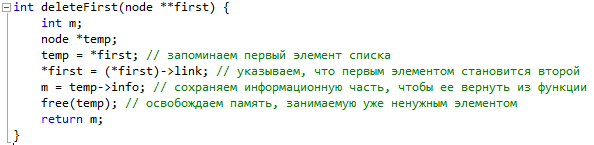
переменная\_указатель = (тип\_указателя)malloc(объем\_в\_байтах)

Так, newNode – указатель на структуру типа node, под которую выделяется память, (node\*) – для значений такого типа будет выделена память, sizeof(node) – размер, требуемый для хранения одной структуры.

Последний элемент добавляется в список так:



Удаление первого элемента списка запрограммировать очень легко:



В данном примере не предусмотрено, что список может не содержать элементов. Вам же следует это предусмотреть.

Оценка выполнения задания 20ХХ\_SAS03 производится следующим образом:

Оценка 5 (отлично): все пункты задания выполнены полностью, программа является работоспособной, исполнитель обоснованно отвечает на контрольные вопросы.

Оценка 4 (хорошо): все пункты задания выполнены полностью, программа является работоспособной, исполнитель хорошо отвечает на контрольные вопросы, но не всегда может обосновать их.

Оценка 3 (удовлетворительно): пункты задания выполнены частично, программа является частично работоспособной, исполнитель затрудняется в ответах на контрольные вопросы.

Оценка 2 (неудовлетворительно): пункты задания выполнены частично, программа не является работоспособной, исполнитель затрудняется в ответах на контрольные вопросы.

Соответствие оценок баллам балльно-рейтинговой системы при оценивании выполнения практической работы:

Оценка 5 (отлично) – 4 балла;

Оценка 4 (хорошо) – 3 балла;

Оценка 3 (удовлетворительно) – 2 балла;

Оценка 2 (неудовлетворительно) – 0 баллов.