**Отчет о выполнении лабораторной работы №1**

**Мурашов Иван Вячеславович, НКАбд-04-23**

1. Объявляем структуру узла, содержащую в себе 3 поля (string, int, double) и ссылку на следующий элемент.

struct Human {  
 string name;  
 int growth;  
 double weight;  
 Human\* next;  
};

1. Реализуем функцию добавления элемента в начало списка. Передаём ссылку на указатель первого элемента и поля нашей структуры (далее «базовые параметры»). Создаём новый объект типа Human, его адрес присваиваем переменной (newNode) Далее “кладём” в функцию базовые параметры, передаваемые в функции. Устанавливаем ссылку нового узла на ‘голову’ списка (pNode) и устанавливаем новый узел как ‘голову’ списка.

void AddFirst(Human\* &pNode, string name, int growth, double weight) {  
 Human\* newNode = new Human;  
 newNode->name = name;  
 newNode->growth = growth;  
 newNode->weight = weight;  
 newNode->next = pNode;  
 pNode=newNode;  
}

1. Реализуем функцию добавления элемента в конец списка. Передаём ссылку на указатель первого элемента и базовые параметры. Создаём новый объект типа Human, его адрес присваиваем переменной (newNode). Далее “кладём” в функцию базовые параметры и устанавливаем ссылку нового узла на ‘голову’ списка (pNode). Если у нас пустой список, то по сути мы создаём 1ый элемент. Else мы ищем последний элемент, после чего кладем в tmp адрес последнего элемента списка.

void AddLast(Human\* &pNode, string name, int growth, double weight) {  
 Human\* newNode = new Human;  
 newNode->name = name;  
 newNode->growth = growth;  
 newNode->weight = weight;  
 newNode->next = pNode;  
 if (pNode == nullptr) {  
 newNode->next = nullptr;  
 pNode = newNode;  
 }  
 else {  
 Human\* tmp = pNode;  
 while (tmp->next != nullptr)  
 tmp = tmp->next;  
 newNode->next = nullptr;  
 tmp->next=newNode;  
 }  
}

1. Реализуем функцию добавления элемента после заданного. Передаём указатель на первый элемент для создания копии внешнего объекта (pNode), строку, по которой будет происходить поиск узла и базовые параметры. Если у нас пустой список, то выводим сообщение об этом. Ищем нужный нам элемент, т.е. устанавливаем начало списка и движемся по списку. Если дошли до последнего элемента и не нашли нужный, выводим соответствующее сообщение.

void AddAfter (Human\* pNode, string Node, string name, int growth, double weight) {  
 if (pNode == nullptr) {  
 cout << "This list is empty\n";  
 return;  
 }  
 Human\* tmp = pNode;  
 while (tmp->next != nullptr and tmp->name != Node)  
 tmp = tmp->next;  
 if (tmp->next == nullptr and tmp->name != Node) {  
 cout << "No element in the list\n";  
 return;  
 }  
 Human\* newNode = new Human;  
 newNode->name = name;  
 newNode->growth = growth;  
 newNode->weight = weight;  
 newNode->next = pNode->next;  
 pNode->next = newNode;  
}

1. Реализуем функцию добавления элемента перед заданным. Передаём указатель на первый элемент для создания копии внешнего объекта (pNode), строку, по которой будет происходить поиск узла и базовые параметры. Если у нас пустой список, то выводим сообщение об этом. Если первый же элемент является искомым, то вызываем функцию AddFirst. Создаём переменные, хранящие ссылки на предыдущий (prev) и последующий (tmp) элементы. Далее цикл для поиска нужного узла (Node). Если дошли до последнего элемента и не нашли нужный, выводим соответствующее сообщение. Если нашли, выполняем стандартные действия, заполняя новый узел и устанавливаем поле next на tmp, а поле next у prev устанавливаем на новый созданный узел.

void AddBefore(Human\* pNode, string Node, string name, int growth, double weight) {  
 if (pNode == nullptr) {  
 cout << "This list is empty\n";  
 return;  
 }  
 if (pNode->name == Node) {  
 AddFirst(pNode, name, growth, weight);  
 return;  
 }  
 Human \*prev = pNode, \*tmp = pNode->next;  
 while (tmp->next != nullptr and tmp->name != Node) {  
 tmp = tmp->next;  
 prev = prev->next;  
 }  
 if (tmp->next == nullptr and tmp->name != Node) {  
 cout << "No element in the list\n";  
 return;  
 }  
 Human\* newNode = new Human;  
 newNode->name = name;  
 newNode->growth = growth;  
 newNode->weight = weight;  
 newNode->next = tmp;  
 prev->next = newNode;  
}

1. Реализуем функцию удаления узла по имени (name). Передаём ссылку на указатель первого элемента и строку, по которой будет происходить поиск узла. Если у нас пустой список, то выводим сообщение об этом. Создаём переменную tmp, устанавливаем её на начало списка. Если первый элемент списка – искомый, то удаляяем его и переопределяем pNode (указатель на первый элемент списка). Если у нас пустой список, то выводим сообщение об этом. Создаём переменную prev и кладём в неё pNode. В tmp передаём pNode->next. Далее ищем нужный элемент. Если дошли до последнего элемента и не нашли нужный, выводим соответствующее сообщение. Если нашли искомый элемент, то переопределяем prev->next и удаляем сам элемент (tmp).

void Del(Human\* &pNode, string Node) {  
 if(pNode == nullptr) {  
 cout << "List is empty\n";  
 return;  
 }  
 Human \*tmp = pNode;  
 if(pNode->name == Node) {  
 pNode = pNode->next;  
 delete tmp;  
 return;  
 }  
 if (pNode->next == nullptr) {  
 cout << "No element in the list\n";  
 return;  
 }  
 Human \*prev=pNode;  
 tmp = pNode->next;  
 while (tmp->next != nullptr and tmp->name != Node) {  
 tmp = tmp->next;  
 prev = prev->next;  
  
 }  
 if(tmp->next == nullptr and tmp->name != Node) {  
 cout << "No element in the list\n";  
 return;  
 }  
 if(tmp->name == Node) {  
 prev->next = tmp->next;  
 delete tmp;  
 return;  
 }  
}

1. Реализуем функцию вывода списка на экран. В качестве параметра передаём указатель на первый элемент. Если у нас пустой список, то выводим сообщение об этом. Устанавливаем tmp на начало списка. Затем выводим данные узла, переходим на следующий элемент (tmp = tmp->next) и так до тех пор, пока не дойдем до последнего элемента.

void Showmen(Human \*pNode) {  
 if (pNode == nullptr) {  
 cout << "This list is empty\n";  
 return;  
 }  
 Human \*tmp = pNode;  
 while(tmp != nullptr) {  
 cout<<"Name: " << tmp->name << "\nGrowth: " << tmp->growth << "\nWeight: " << tmp->weight << endl << endl;  
 tmp = tmp->next;  
 }  
}

1. В функции main реализуем сценарий работы со списком, использующий разработанный инструментарий. Создаём список list1. Далее последовательно выполняем ранее реализованные функции. Добавляем в начало (“Tom”), добавляем в начало (“Lisa”), добавляем после Tomа (“Fred ”), добавляем перед Fredом (“Jeff ”), удаляем (“Lisa ”), добавляем в конец (“Mary ”).

int main() {  
 Human\* list1 = nullptr;  
 AddFirst(list1, "Tom", 175, 70.55);  
 Showmen(list1);  
 cout<<"-----------------------------------------------------------------------"<<endl << endl;  
 AddFirst(list1, "Lisa", 150, 59.64);  
 Showmen(list1);  
 cout<<"-----------------------------------------------------------------------"<<endl << endl;  
 AddAfter(list1, "Tom", "Fred", 189, 100.38);  
 Showmen(list1);  
 cout<<"-----------------------------------------------------------------------"<<endl << endl;  
 AddBefore(list1, "Fred", "Jeff", 140, 56.49);  
 Showmen(list1);  
 cout<<"-----------------------------------------------------------------------"<<endl << endl;  
 Del(list1, "Lisa");  
 Showmen(list1);  
 cout<<"-----------------------------------------------------------------------"<<endl << endl;  
 AddLast(list1, "Mary", 162, 51.26);  
 Showmen(list1);  
 cout<<"-----------------------------------------------------------------------"<<endl << endl;  
  
 return 0;  
}