Списки, очереди. Списки, очереди. Как списки и очереди связаны вообще с программированием. Есть списки на отчисление из-за не сданной проги , есть очереди в деканат за ведомостичкой. Но как это вообще связано с программированием!?

Ну раз не понятно давай разбираться. Начнем с того что список и очередь это в принципе одно и тоже. А почему они тогда придумали 2 название на одно и тоже? Они что, дураки? Так учить много, а тут еще и два слова на одну и туже конструкцию!  
  
Ну,2 определения для одного и того же образовалась случайно. Для того чтобы студент мог ассоциировать эту структуру данных с чем-то из реальной жизни придумали 2 ярчайших примера. Список в котором новый элемент добавляется в конец и очередь в которой новый покупатель становится в ее самый конец. Разные преподаватели приводили в качестве примера или список или очередь из-за чего и возникла такая путаница.

С этим разобрались. А что это вообще за структура данных такая? Где используется и зачем мне нужно сдавать по ней экзамен?

Структурой данных называют обычно какой-то определенный способ хранения данных. Программисты пришли к удивительному открытию – если хранить данные абы как , то их сложнее найти. Это как носки в комнате. Один на люстре , второй в раковине, третий участвует в заварке чая , а четвертый … а кто нибудь его вообще видел? И информацию решили структурировать и хранить в определенном порядке(Прям как в жизни). Очередь используется очень часто. Самое её банальное применение – это(вы не поверите) онлайн очередь! Очередь в больнице,кинотеатре, McDonald’s и тд. Всем она нужна.

Теперь давайте посмотрим как выглядит этот зверь в c++

#include "pch.h"

#include <iostream>

using namespace std;

struct queue

{

int info;

queue \* next;

} \*head=nullptr;

queue\* Add(queue \*start,int information)

{

queue \*t = start;

if (t==nullptr)

{

t = new queue();

t->info = information;

t->next = nullptr;

return t;

}

while (t->next!=nullptr)

{

t = t->next;

}

t->next = new queue();

t->next->next = nullptr;

t->next->info = information;

return start;

}

queue\* DellAll(queue \*start)

{

queue \*toDel;

while (start!=nullptr)

{

toDel = start;

start = start->next;

delete toDel;

}

cout << endl << "Sector clear!";

return start;

}

void view(queue \*start)

{

while (start != nullptr)

{

cout << start->info << " ";

start = start->next;

}

}

void main()

{

for(int i=0;i<5;i++)

head = Add(head,i);

view(head);

DellAll(head);

}

Воу,воу,воу! Полегче. Это вообще что? Объясняй давай , я тут только int с main знаю!

Ок. Давай по порядку разберем эту зверюгу и поймем, что здесь нет ничего сложного.

Встречает нас такая штука:  
  
struct queue

{

int info;

queue \* next;

} \*head=nullptr;

Давайте разбираться.

struct queue – этой строчкой мы создаем наш пользовательский тип данных queue.

Типов данных ты знаешь много int,double,char и другие. Их создали программисты для своих целей. А ты создаешь свой тип данных queue. Это уже твой личный тип данных и хранить он будет то , что ты ему скажешь хранить.

А что он вообще будет хранить? Хранить он будет некую информацию (пусть будет типа int ) и указатель на следующий элемент очереди. Выглядеть это будет вот так  
  
struct queue

{

int info;

queue \* next;

}

Класс , вот ты так это написал , а как это представить в воображении? Я советую представлять вот так:

Какое-то число

Адрес на след. элемент

Какое-то число

Адрес на след. элемент

Ладно, притворимся что это понятно, что дальше? А дальше какой то \*head за скобкой.

struct queue

{

int info;

queue \* next;

} \*head=nullptr;

И почему он не в скобке? Не пустили? Плохо себя вел? Нет! Этот одинокий отшельник называется ГЛОБАЛЬНЫЙ указатель!

Ок,если это указатель то на что он указывает? Где тип данных на который он указывает?

Этот указатель указывает на queue. Как так произошло? Ну,если коротко то когда наша программа компилируется то для этого указателя head происходят некоторая магия. Во время компиляции убирается слово struct. Теперь выглядит наш код вот так

queue

{

int info;

queue \* next;

} \*head=nullptr;

Далее убирается все что в фигурных скобках и остается вот что

queue \*head=nullptr;

Вот, уже знакомая запись.

Ок,глобальный указатель есть. А зачем он нам вообще?

А вопрос вообще-то очень хороший! Для функций которые мы написали для работы со списком он действительно не нужен! Я его тут вставил только потому что в методичках он там стоит и все! И преподы требуют чтобы он там был , а вот зачем он там никто рассказывать не хочет.

Ну раз никто рассказывать не хочет , то ты расскажи о великий и всезнающий Рудик!

Ну ладно ☺. Расскажу. Глобальный указатель удобен тем , что его можно вызвать из любого места программы благодаря чему нам не нужно передавать указатель на вершину стека в аргументах функции. Если использовать этот указатель правильно , то код будет выглядеть вот так:

#include "pch.h"

#include <iostream>

using namespace std;

struct queue

{

int info;

queue \* next;

} \*head=nullptr;

void Add(int information)

{

if (head == nullptr)

{

head = new queue();

head->info = information;

head->next = nullptr;

return;

}

queue \*t = head;

while (t->next!=nullptr)

{

t = t->next;

}

t->next = new queue();

t->next->next = nullptr;

t->next->info = information;

}

void DellAll()

{

queue \*toDel;

while (head !=nullptr)

{

toDel = head;

head = head->next;

delete toDel;

}

cout << endl << "Sector clear!";

}

void view()

{

queue \*t = head;

while (t != nullptr)

{

cout << t->info << " ";

t = t->next;

}

}

void main()

{

for(int i=0;i<5;i++)

Add(i);

view();

DellAll();

}

Результат тот же. Кода меньше и с возвратом мучатся не нужно.

Ладно, с этим разобрались. Идем дальше.

queue\* Add(queue \*start,int information)

{

queue \*t = start;

if (t==nullptr)

{

t = new queue();

t->info = information;

t->next = nullptr;

return t;

}

while (t->next!=nullptr)

{

t = t->next;

}

t->next = new queue();

t->next->next = nullptr;

t->next->info = information;

return start;

}

Классная функция. Она заполняет нашу очередь. Очередь заполняется очень просто – новый элемент добавляется в конец. Для начала проверим есть ли элементы в нашей очереди вообще.

queue \*t = start; - создаем указатель t который ставим на вершину очереди. С помощью него мы будем идти до конца списка чтобы добавить новый элемент в конец.

if (t==nullptr)

{

t = new queue();

t->info = information;

t->next = nullptr;

return t;

}

if (t==nullptr) Проверяем – если наш указатель ни на что не указывает , значит список пуст и нужно добавить в него первый элемент.

{

t = new queue();//Выделяем память под новый элемент

t->info = information; //Заполняем поле info этого элемента

t->next = nullptr; //Новый элемент всегда указывает на nullptr так как он последний

return t; //Возвращаем указатель на нашу вершину

}

Если элементы в нашей очереди есть,то нам нужно дойти до последнего элемента чтобы его полю next присвоить адрес на новый элемент.

while (t->next!=nullptr) //Пока t не дойдет до последнего элемента t все время идёт

{

t = t->next;

}

После того как выполнится этот цикл мы можем смело сказать что t указывает на последний элемент , а это значит что теперь мы можем заняться добавлением нашего нового элемента в конец

t->next = new queue();//Поле next последнего элемента теперь указывает на новый элемент

t->next->next = nullptr; //Заполняем Поле next последнего элемента

t->next->info = information; //Заполняем Поле info последнего элемента

return start; //Возвращаем вершину

Ну вот. Функцию добавления разобрали. Ничего сложного в ней нет. Идём дальше.

queue\* DellAll(queue \*start)

{

queue \*toDel;

while (start!=nullptr)

{

toDel = start;

start = start->next;

delete toDel;

}

cout << endl << "Sector clear!";

return start;

}

Это функция нужна для удаления всех элементов очереди. Давайте разбираться.

queue \*toDel; //Этот указатель будет указывать на элемент который мы будем удалять

while (start!=nullptr) //Идем пока не дойдем до конца

{

toDel = start; //Ставим указатель toDel на элемент который будем удалять

start = start->next; //Переходим на след. элемент

delete toDel; //Освобождаем память от элемента (А то он слишком жирный , памяти на него не напасёшься)

}

cout << endl << "Sector clear!"; // Говорим пользователю что все ОК

return start; //Возвращаем указатель на вершину(Он будет всегда равен nullptr)

Ну , эта вроде полегче была. Перейдём к десерту. Просмотр списка.

void view(queue \*start)

{

while (start != nullptr)

{

cout << start->info << " ";

start = start->next;

}

}

Ну тут вообще все просто, просто идем до конца и выводим значение элемента на котором стоим через cout. И это были те самые страшные очереди?  
  
  
ДА,это основные операции над очередью и ничего сложного в них нет. Осталось рассмотреть только их вызов.

void main()

{

for(int i=0;i<5;i++)

head = Add(head,i); //Создаем 5 элементов в очереди

view(head); // Просматриваем очередь

DellAll(head); // Удаляем очередь

}

Ну вот и все. Тот страшный не понятный код превратился во вполне читаемый текст который ты теперь надеюсь понимаешь.

#include "pch.h"

#include <iostream>

using namespace std;

struct queue

{

int info;

queue \* next;

} \*head=nullptr;

queue\* Add(queue \*start,int information)

{

queue \*t = start;

if (t==nullptr)

{

t = new queue();//Выделяем память под новый элемент

t->info = information; //Заполняем поле info этого элемента

t->next = nullptr; //Новый элемент всегда указывает на nullptr так как он последний

return t; //Возвращаем указатель на нашу вершину

}

while (t->next!=nullptr) ) //Пока t не дойдет до последнего элемента t все время идёт

{

t = t->next;

}

t->next = new queue();//Поле next последнего элемента теперь указывает на новый элемент

t->next->next = nullptr; //Заполняем Поле next последнего элемента

t->next->info = information; //Заполняем Поле info последнего элемента

return start; //Возвращаем вершину

}

queue\* DellAll(queue \*start)

{

queue \*toDel; //Этот указатель будет указывать на элемент который мы будем удалять

while (start!=nullptr) //Идем пока не дойдем до конца

{

toDel = start; //Ставим указатель toDel на элемент который будем удалять

start = start->next; //Переходим на след. элемент

delete toDel; //Освобождаем память от элемента (А то он слишком жирный , памяти на него не напасёшься)

}

cout << endl << "Sector clear!"; // Говорим пользователю что все ОК

return start; //Возвращаем указатель на вершину(Он будет всегда равен nullptr)

}

void view(queue \*start)

{

while (start != nullptr) // Идем до конца

{

cout << start->info << " ";//Выводим значение элемента на котором находимся

start = start->next; //Идем на след. элемент

}

}

void main()

{

for(int i=0;i<5;i++)

head = Add(head,i); //Добавляем элемент

view(head); //Просматриваем элементы

DellAll(head); //Удаляем элементы

}

НА этом у меня вроде бы все.

Внимание!!!

Спасибо за внимание!!!