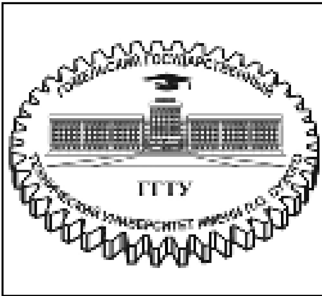

Министерство образования Республики Беларусь

Учреждение образования
«Гомельский государственный технический
университет имени П. О. Сухого»

Кафедра «Информатика»

ЯЗЫК ПРОГРАММИРОВАНИЯ С/С++

ПРАКТИКУМ
по дисциплине «Программирование» для студентов специальности 1-40 04 01 «Информатика и технологии программирования» дневной формы обучения

Электронный аналог печатного издания

Гомель 2018

Библиотека ГГТУ им. П.О. Сухого

УДК 004.43(075.8)
ББК 32.973.2я73
Я41

Рекомендовано к изданию научно-методическим советом
факультета автоматизированных и информационных систем
ГГТУ им. П. О. Сухого
(*протокол № 11 от 27.06.2017 г.*)

Составитель *Г. П. Косинов*

Рецензент: зав. каф. «Вычислительная математика и программирование» ГГУ им. Ф. Скорины, доцент, канд. физ.-мат. наук, доц. *Д. С. Кузьменков*

Я41 **Язык** программирования C/C++ : практикум по дисциплине «Программирование» для студентов специальности 1-40 04 01 «Информатика и технологии программирования» днев. формы обучения / сост. Г. П. Косинов. – Гомель : ГГТУ им. П. О. Сухого, 2018. – 81 с. – Систем. требования: PC не ниже Intel Celeron 300 МГц ; 32 Mb RAM ; свободное место на HDD 16 Mb ; Windows 98 и выше ; Adobe Acrobat Reader. – Режим доступа: https://elib.gstu.by. – Загл. с титул. экрана.

ISBN 978-985-535-368-4.

Содержит задания для лабораторных работ по дисциплине «Программирование» и краткие теоретические сведения для их выполнения.

Для студентов специальности 1-40 04 01 «Информатика и технологии программирования» дневной формы обучения.

УДК 004.43(075.8)
ББК 32.973.2я73

ISBN 978-985-535-368-4

© Косинов Г. П., составление, 2018
© Учреждение образования «Гомельский государственный технический университет имени П. О. Сухого», 2018

Библиотека ГГТУ им. П.О.Сухого

Оглавление

Глава 1. Списки данных ... 5
1.1. Реализация однонаправленного списка ... 6
1.2. Создание однонаправленного списка ... 6
1.3. Добавление и удаление элемента из списка ... 6
1.4. Поиск элемента в однонаправленном списке ... 8
1.5. Удаление однонаправленного списка ... 9
1.6. Двунаправленные списки ... 9
Практическая часть ... 10
Глава 2. Списковые конструкции ... 13
2.1. Стек ... 13
2.2. Очередь ... 15
2.3. Кольцевой буфер ... 17
Глава 3. Деревья ... 19
3.1. Бинарные деревья ... 20
3.2. Реализация дерева ... 21
3.3. Построение дерева ... 21
3.4. Вставка нового элемента в дерево ... 22
3.5. Способы обхода дерева ... 23
3.6. Поиск элемента в дереве ... 24
3.7. Удаление узла по ключу ... 25
3.8. Удаление дерева ... 28
Практическая часть ... 28
Глава 4. Препроцессор ... 30
4.1. Трехзнаковые последовательности ... 30
4.2. Макроопределение и макрорасширение ... 31
4.3. Включение файлов ... 31
4.4. Условная компиляция ... 32
4.5. Заранее определенные имена ... 33
Глава 5. ЯЗЫК С++ ... 33
5.1. Основные концепции (объектно-ориентированное программирование) ... 36
5.2. Инкапсуляция ... 37
Практическая часть ... 38
Глава 6. Наследование классов ... 40
6.1. Дружественные функции ... 43
6.2. Множественное наследование ... 44
Практическая часть ... 45

Библиотека ГГТУ им.П.О.Сухого

Глава 7. Пространства имен и исключительные ситуации 48
7.1. Пространство имен 48
7.2. Директивы using 50
7.3. Объявление пространства имен и их членов 50
7.4. Глобальное пространство имен 51
7.5. Обработка ошибок и исключений 51
Глава 8. Полиморфизм 53
8.1. Перегрузка функций 55
8.2. Перегрузка конструктора 57
Глава 9. Стандартная библиотека шаблонов (STL) 58
9.1. Контейнеры 58
9.2. Алгоритмы 59
9.3. Итераторы 59
9.4. Другие элементы библиотеки STL 60
9.5. Строки в C++ 60
9.6. Контейнеры STL 63
9.7. Контейнеры последовательности 63
9.8. Ассоциативные контейнеры 64
9.9. Контейнеры-адаптеры 64
9.10. Класс vector 65
9.11. Список методов для работы с вектором 65
9.12. Класс stack 67
9.13. Класс queue 68
Практическая часть 68
Глава 10. Основы программирования под Windows 71
Глава 11. Обработка событий 74
Практическая часть 75
Глава 12. Графические приложения Windows 76
Практическая часть 78

Библиотека ГГТУ им. П.О. Сухого

ГЛАВА 1. СПИСКИ ДАННЫХ

В программировании существует множество способов решения одной и той же задачи. И в каждом случае алгоритмы решения не бывают идеальными, всегда если выигрываем одном, то проигрываем в другом. В данном соревновании нет единственно верного решения. Практически всегда приходится идти на компромиссы.

Абстрактная структура данных – список – представляет из себя гибкую модель, с которой легко производить операции изменения – такие как вставка элементов в список, удаление элементов из списка, операции объединения и разделения списков и т. д.

Реализовать список в языке программирования С++ можно с помощью массива и с помощью указателей. Каждый из методов имеет свои преимущества и свои недостатки. Мы рассмотрим реализацию списков только с использованием указателей.

Графически реализацию списка можно изобразить следующим образом (рис. 1.1).

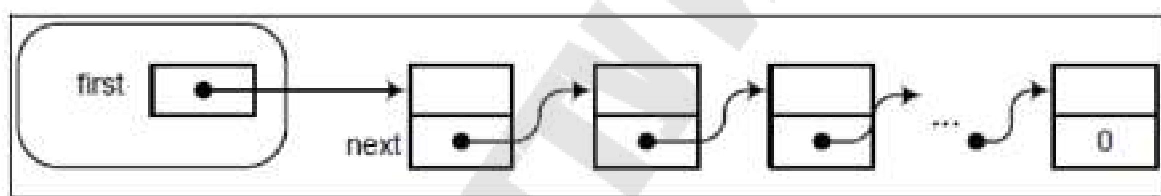

Рис. 1.1. Реализация списка

При такой реализации списка каждый элемент списка состоит из информационной и указательной частей. Информационная часть может состоять из одного и более полей, указательная часть всегда состоит из одного указателя и его тип совпадает с типом самой структуры.

Реализация списка с помощью указателей, в отличие от реализации списка с помощью массива, не требует непрерывного участка памяти и заранее зарезервированного количество памяти. Связывание элементов списка происходит с помощью указателей, т. е. соседний элемент ссылается на следующий элемент и т. д. до последнего элемента, который имеет пустой указатель.

Такая реализация списка позволяет проводить операции удаления вставки и замены элементов в списке с наименьшим количеством операций с памятью. Однако для такой реализации списка требуется дополнительная память для хранения ссылки на следующий элемент списка.

Библиотека ГГТУ им. П. О. Сухого

1.1. Реализация однонаправленного списка

Для начала описания алгоритмов определим структуру списка, которую будем использовать в дальнейшем для демонстрации работы алгоритмов.

```
struct List {
int info;
List *next;
};
```

Структура, с которой мы будем работать в дальнейшем, состоит из двух полей. Поле info имеет тип int и выполняет роль информационной части элемента списка. Поле next типа указателя на такую же структуру по типу играет роль указателя на следующий элемент списка.

1.2. Создание однонаправленного списка

Для создания списка в программе необходимо создать указатель на первый элемент, который будет называться «головой списка». И присвоить ему значение NULL. Таким образом, мы создали пустой список, который не содержит ни одного элемента и его размерность равна 0.

```
List * Head = NULL;
```

1.3. Добавление и удаление элемента из списка

Операция добавления нового элемента в список осуществляется по средству изменения ссылок. Операция добавления элемента в однонаправленный список осуществляется после элемента, на который установлен указатель. Для этого необходимо ссылку на следующий элемент из текучего элемента скопировать в новый элемент, а в текущем элементе записать в ссылку на следующий элемент ссылку на новый элемент.

Алгоритмы добавления в начало и любое другое место списка отличаются друг от друга. Поэтому в функции, реализующей данную операцию, осуществляется проверка, в какое место добавляется новый элемент. Далее реализуется соответствующий алгоритм добавления (рис. 1.2).

Библиотека ГГТУ им. П.О. Сухого

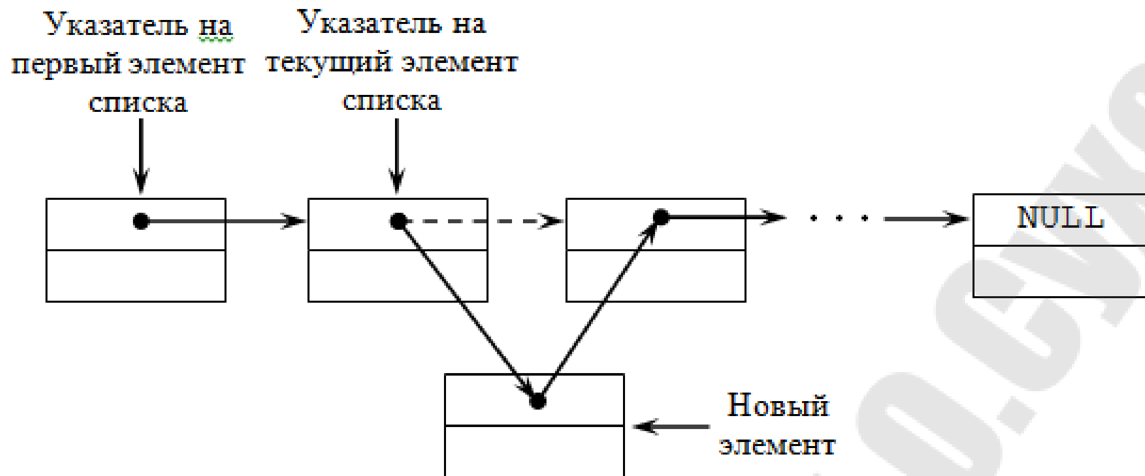

Рис. 1.2. Реализация добавления нового элемента в список

Реализовать данный алгоритм на языке C++ можно следующим образом:

```
List* add(List** list, int data, int position)
{
 position--;
 int index = 0;
 List *item = createList(data);
 if (*list == NULL)
 {
 *list = item;
 }
else {
List *Current = *list;
for (int i = 1; i < position && Current->next != NULL; i++)
Current = Current->next;
if (position == 0)
 {
 item->next = *list;
 *list = item;
 }
 else {
  if (Current->next != NULL)
  item->next = Current->next;
  Current->next = item;
 }
```


Библиотека ГГТУ им. П.О.Сухого

```
}
return *list;
}
```

Из динамических структур можно удалять достаточно быстро элементы, так как для этого достаточно изменить значения адресных полей. Операция удаления элемента однонаправленного списка осуществляет удаление элемента, на который установлен указатель текущего элемента. После удаления указатель текущего элемента устанавливается на предшествующий элемент списка или на новое начало списка, если удаляется первый.

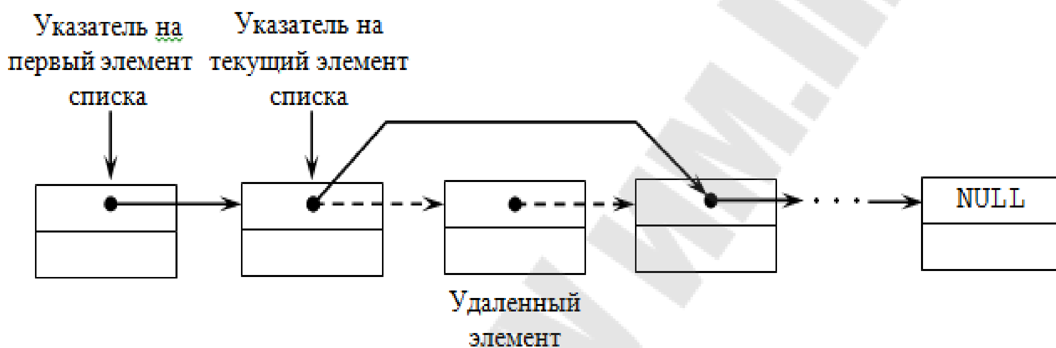

Рис. 1.3. Реализация удаления элемента из списка

Алгоритмы удаления первого и последующих элементов списка отличаются друг от друга. Поэтому в функции, реализующей данную операцию, осуществляется проверка, какой элемент удаляется. Далее реализуется соответствующий алгоритм удаления (рис. 1.3).

1.4. Поиск элемента в однонаправленном списке

Операция поиска элемента в списке заключается в последовательном просмотре всех элементов списка до тех пор, пока текущий элемент не будет содержать заданное значение или пока не будет достигнут конец списка. В последнем случае фиксируется отсутствие искомого элемента в списке (функция вернет пустую ссылку).

```
List* getByInfo(List* list, int info) {
List* Current = list;
List* result = NULL;
while (Current->next != NULL && Current->info != info) {
Current = Current->next;
```


Библиотека ГГТУ им. П. О. Сухого

```
}
if (Current->info == info) result = Current;
return result;
}
```

1.5. Удаление однонаправленного списка

После окончания работы со списком необходимо очистить динамическую память. Для этого нужно удалить все существующие ссылки. На языке C++ это лучше всего сделать с помощью рекурсивной функции.

Операция удаления списка заключается в освобождении динамической памяти. Для данной операции организуется функция, в которой нужно переставлять указатель на следующий элемент списка до тех пор, пока указатель не станет равен NULL, т. е. не будет достигнут конец списка. Реализуем рекурсивную функцию.

```
void dellAll(List** Head)
{
 if (*Head != NULL)
 {
  dellAll((&(*Head)->next));
  (*Head) = NULL;
 }
}
```

Таким образом, однонаправленный список имеет только один указатель в каждом элементе. Это позволяет минимизировать расход памяти на организацию такого списка. Одновременно это позволяет осуществлять переходы между элементами только в одном направлении, что зачастую увеличивает время, затрачиваемое на обработку списка. Например, для перехода к предыдущему элементу необходимо осуществить просмотр списка с начала до элемента, указатель которого установлен на текущий элемент.

1.6. Двунаправленные списки

В однонаправленном списке мы имеем возможность выполнить проход по списку только в одном направлении, однако для некоторых задач такой организации данных недостаточно, и необходимо дви-

Библиотека ГГТУ им. П.О. Сухого

гаться по элементам в двух направлениях. И для решения такой задачи в структуру добавляется указатель не только на следующий элемент в списке, но и на предыдущий элемент (рис. 1.4).

Рис. 1.4. Реализация двунаправленного списка

Практическая часть

Односвязные списки

Реализовать структуру односвязный список на основе пары–значение (указатель и число).

Во всех вариантах предусмотреть динамическое выделение памяти и освобождение неиспользуемых участков. Предусмотреть некорректный ввод и обработку ошибок при вводе. Во всех задачах перед обработкой выводить исходный список.

1. Составить список из N чисел. Проверить его на наличие одинаковых элементов. Одинаковые элементы перенести в начало списка.

2. Дан односвязный список. Сформировать следующим образом: первый элемент – максимальный, второй – минимальный, третий – n-1 по максимальности и т. д. То есть идет чередование: максимальный, минимальный, 2-й по максимальности, 2-й по минимальности и т. д.

3. Составить однонаправленный список из N чисел. Проверить его на наличие уникальных элементов. Одинаковые элементы удалить и перенести их в новый список.

4. Создать однонаправленный список из случайных целых чисел. Удалить из этого списка все элементы, находящиеся между максимальным и минимальным элементами. Сформировать новый список из элементов, которых нет между максимальным и минимальным элементом. Например, 2, 5, 28, 29, 0, 3, 3 => 3, 3

5. Дан односвязный список. Определить его длину. Найти максимальный элемент, минимальный элемент, среднее арифметическое. Вывести их. В случае если максимальный элемент больше 20, заменить все значения меньше среднего арифметического нулями, иначе вывести сообщение об этом.

Библиотека ГГТУ им. П.О.Сухого

6. Создать двунаправленный список из случайных чисел. Определить среднее арифметическое значение всех элементов. Сформировать два новых списка, поместив в них значения меньшие среднего арифметического и больше соответственно.

7. Даны два односвязных списка. Первый отсортирован по возрастанию, а второй по убыванию. Сформировать новый список, отсортированный по возрастанию.

8. Дан односвязный список. Найти в нем максимальное произведение 3-х чисел и вывести его. Сформировать новый список из этих 3-х чисел и его тоже вывести.

9. Составить однонаправленный список из N чисел. Сформировать новый список из элементов исходного списка, отсортированный по убыванию, в котором все числа больше среднего арифметического.

10. В односвязном списке найти минимальный по модулю элемент. Из элементов, больших по модулю чем он, сформировать новый список.

11. В односвязном списке найти наибольший нечетный элемент. Далее трижды осуществите циклический сдвиг влево элементов, стоящих справа от найденного максимума, и один раз сдвиг элементов вправо, стоящих слева от найденного максимального нечетного элемента.

12. Организовать односвязный список так, чтобы сначала шли нулевые элементы, а затем все остальные.

13. Организовать односвязный список случайными числами, вывести его на экран. Найти самую длинную последовательность чисел, упорядоченную по возрастанию. Вывести ее на экран. Если таких последовательностей несколько (самых длинных с одинаковой длиной), то вывести их все.

14. Даны два односвязных списка с различным количеством элементов. Перераспределить их элементы так, чтобы в первом списке были наименьшие числа из двух списков, а во втором - наибольшие.

15. Дан список из 50 элементов, значения которых формируются функцией и лежат в диапазоне от –50 до 49 включительно. Требуется из одного списка скопировать в другой значения в диапазоне от –5 до 5 включительно и подсчитать их количество.

16. Имеется односвязный список, содержащий числа от 0 до 49 включительно. Требуется исключить из него все элементы, значения которых меньше 15 и сформировать из них новый список.

17. Сдвинуть элементы односвязного списка в указанном направлении (влево или вправо) и на указанное число шагов. Освободившиеся ячейки заполнить нулями. Выводить список после каждого шага.

Библиотека ГГТУ им. П.О.Сухого

18. Заполнить односвязный случайными положительными и отрицательными числами таким образом, чтобы все числа по модулю были разными. Это значит, что в списке не может быть ни только двух равных чисел, но не может быть двух равных по модулю. В полученном списке найти наибольшее по модулю число, вывести его и числа справа больше найденного.

19. Заполнить список случайными положительными и отрицательными целыми числами. Вывести его на экран. Удалить из списка все отрицательные элементы и снова вывести.

20. Создать три однонаправленных списка из целых чисел. Сформировать четвертый список из элементов, которые есть только в одном из трех списков, но нет в двух других.

21. Создать два однонаправленных списка из целых чисел. Сформировать третий список из элементов, которые есть в обоих.

22. Даны два списка, значения в которых отсортированы по возрастанию. Сформировать третий список из первых двух, значения первой половины которого будут отсортированы по возрастанию, а второй по убыванию.

23. Из элементов одного односвязного списка сформировать два списка, в первом из которых будут элементы больше среднего арифметического исходного массива, а во втором, соответственно, больше среднего арифметического исходного массива.

24. Создать список из случайных положительных и отрицательных целых чисел. Образовать из него два однонаправленных списка, первый должен содержать значения до первого отрицательного числа, а второй – после первого положительного.

25. Создать однонаправленный список из случайных целых чисел и преобразовать его в два списка. Первый должен содержать только положительные числа, а второй – только отрицательные. Затем удалить элементы, по модулю меньшие 5.

26. Описать функцию, которая вставляет в однонаправленный список *A* за первым вхождением элемента *B* все элементы списка *C*, если *B* входит в *A*.

27. Из двух однонаправленных списков сформировать новый следующим образом: сперва записать элементы первого списка, которые являются палиндромами (число из одной цифры – палиндром во все стороны), затем – отрицательные по значению элементы второго списка.

Библиотека БГУИР

28. Среди элементов односвязного списка найти и вывести последовательность, сумма которой максимальна. В найденной последовательности определить четность каждого элемента (вывести да/нет), и если элемент четный, то заменить его нулем.

ГЛАВА 2. СПИСКОВЫЕ КОНСТРУКЦИИ

2.1. Стек

Стек представляет из себя однонаправленный список, организованный по принципу FILO (first-in-last-out). В голове стека хранится адрес на последний добавленный в стек элемент. Для понимания работы стека можно воспользоваться примерами из жизни, такие как колода карт при игре в покер, стопка тарелок на кухне, оружейный магазин. Во всех этих примерах производить операции можно только с вершиной.

Для добавления нового элемента в вершину стека необходимо в адресную часть элемента записать значение вершины, после изменить значение вершины на адрес нового элемента (рис. 2.1).

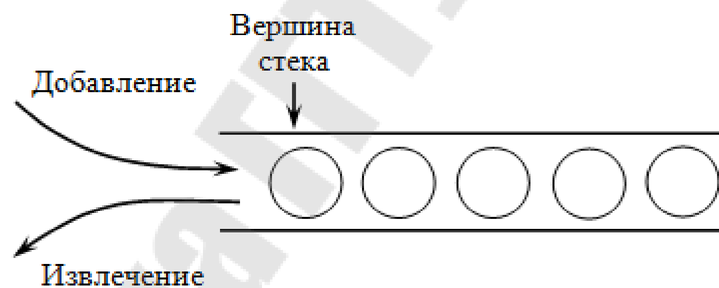

Рис. 2.1. Добавление нового элемента

```
struct stack {
int info;
stack *next;
};
stack* newStackItem(int info) {
stack* newItem = new (stack);
newItem->info = info;
newItem->next = NULL;
return newItem;
}
```


Библиотека ГГТУ им. П.О. Сухого

```
stack* push(stack**head, int info) {
stack *Item = newStackItem(info);
if (*head != NULL) {
Item->next = *head;
*head = Item;
}
else {
*head = Item;
}
return *head;
}
```

Удаление элемента из стека производится с вершины. Для этого берется указатель на следующий элемент у первого элемента и записывается в указатель на вершину списка. Таким образом получается, что мы изменили значение вершины списка на второй элемент стека (рис. 2.2).

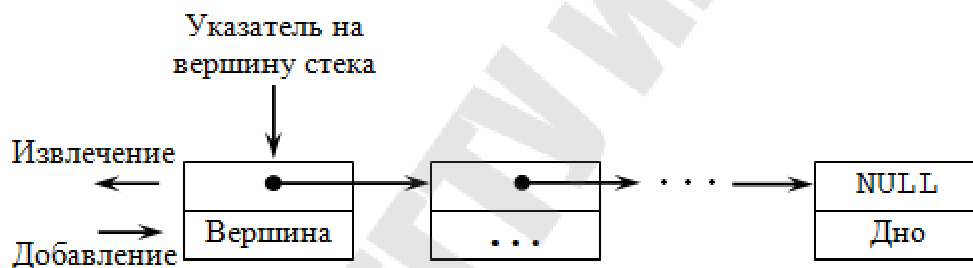

Рис. 2.2. Удаление элемента

```
stack* pop(stack**head) {
stack *Current = NULL;
if (*head != NULL) {
Current = *head;
*head = (*head)->next;
}
return Current;
}
```

Операция удаления стека заключается в освобождении динамической памяти. Для этого необходимо удалить все элементы списка.

```
void delSteck(stack**head) {
if (*head != NULL) {
delSteck( &((*head)->next) );
```


Библиотека ГГТУ им.П.О.Сухого

```
delete *head;
*head = NULL;
}
}
```

2.2. Очередь

Очередь – это однонаправленный линейный список, где новые элементы добавляются в конец очереди, а удаляются элементы с начала списка (FIFO first-in-first-out). Операторы, выполняемые над очередью, аналогичны операциям над стеком, с отличием в том, что в стеке элементы удаляются и добавляются в начало, а в очереди удаление происходит с начала, а добавление новых элементов в конец.

Добавление новых элементов производится в конец очереди. Для этого необходимо взять последний элемент и записать в него ссылку на новый элемент. В итоге новый элемент будет добавлен в наш список (рис. 2.3).

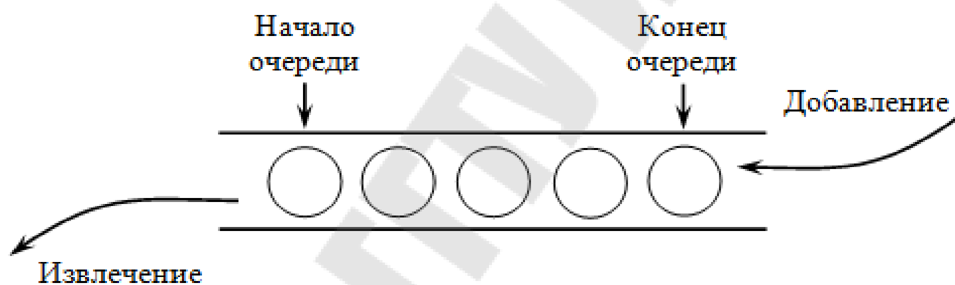

Рис. 2.3. Реализация очереди

```
struct query
{
int info;
query* next;
};
query* newQueryItem(int info) {
query* newItem = new (query);
newItem->info = info;
newItem->next = NULL;
return newItem;
}
query* push(query**head, int info) {
query* item = newQueryItem(info);
```

Библиотека ГГТУ им. П.О.Сухого

```
if (*head != NULL) {
query* Current = *head;
while (Current->next != NULL) {
Current = Current->next;
}
Current->next = item;
}
else {
*head = item;
}
return *head;
}
```

Удаление элементов из очереди происходит также, как и в стеке с начальной позиции. Для этого берется первый элемент в списке и его ссылку на следующий элемент записывают в указатель на вершину списка. Таким образом вершина хранит указатель на второй элемент списка.

```
query* pop(query**head) {
query *Current = NULL;
if (*head != NULL) {
Current = *head;
*head = (*head)->next;
}
return Current;
}
```

Операция удаления очереди выполняется также, как и в стеке. Она заключается в освобождении динамической памяти. Для этого необходимо удалить все элементы списка.

```
void delQuery(query**head) {
if (*head != NULL) {
delQuery(&((*head)->next));
delete *head;
*head = NULL;
}
}
```


Библиотека ГГТУ им. П.О.Сухого

2.3. Кольцевой буфер

Кольцевой буфер – представляет из себя линейный список, который по внутренней организации похож на стек, но с отличием в том, что последний элемент списка хранит ссылку на вершину списка. Кольцевой список организован по принципу FIFO (first-in-first-out). Добавление и удаление элементов кольцевого буфера производится из начала списка.

Добавление нового элемента в кольцевой буфер производится в начало и выполняется по следующему алгоритму. Добавляемый элемент в кольцевой буфер становится вершиной, в него же записывается в качестве следующего значение изначальное значение вершины списка, после этого в списке берется последний элемент списка и в него записывается новое значение вершины списка.

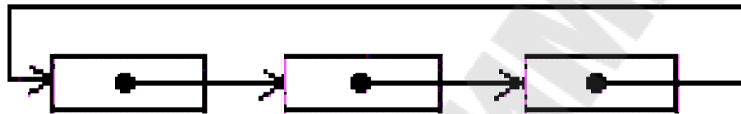

Рис. 2.4. Добавление нового элемента в кольцевой буфер

```
struct RingBuffer
{
int info;
RingBuffer *next;
};
RingBuffer* push(RingBuffer**head, int info) {
RingBuffer *Item = newRingBufferItem(info);
if (*head != NULL) {
RingBuffer *Current = *head;
while (Current->next != *head)
{
Current = Current->next;
}
Item->next = *head;
*head = Item;
Current->next = Item;
}
else {
*head = Item;
Item->next = *head;
}
```


Библиотека ГГТУ им. П.О. Сухого

```
return *head;
}
```

Удаление элементов из кольцевого буфера производится из начала списка. Для этого нужно изменить значение вершины списка на значение следующего элемента, который идет после вершины. После берется последний элемент кольцевого буфера, и его ссылка на следующий элемент меняется на новую вершину.

```
RingBuffer* pop(RingBuffer**head) {
RingBuffer *Result = NULL;
RingBuffer *Current = NULL;
if (*head != NULL) {
Current = (*head)->next;
while (Current->next != *head) {
Current = Current->next;
}
Result = Current->next;
*head = (*head)->next;
Current->next = *head;
}
return Result;
}
```

Операция удаления кольцевого буфера заключается в освобождении динамической памяти. Для этого необходимо удалить все элементы списка.

```
void delRingBuffer(RingBuffer**head) {
RingBuffer*temp = *head;
RingBuffer*Current1 = temp->next;
RingBuffer*Current2 = temp->next;
while (Current1 != temp) {
Current2 = Current1->next;
delete Current1;
Current1 = Current2;
}
delete *head;
*head = NULL;
}
```


Библиотека ГГТУ им. П.О. Сухого

ГЛАВА 3. ДЕРЕВЬЯ

Рассмотренные ранее структуры данных (массивы и списки) имеют линейную структуру. Порядок следования и выборки элементов таких структур имеет линейный характер и соответствует порядку расположения элементов в памяти: один за другим без каких-либо промежутков. Адрес элемента соответствует его положению и определяется индексом – порядковым номером элемента в последовательности размещения.

Существуют структуры, которые не допускают подобной «линеаризации»: их невозможно «вытянуть в линию» и для их изображения необходима плоскость. С точки зрения организации данных это дает разнообразие вариантов размещения одного и того же набора данных, а также различные варианты обхода одной и той же структуры. Порядок следования и выборки элементов таких структур может не соответствовать порядку расположения элементов в памяти. Одним из примеров таких структур являются деревья.

Дерево – структура данных, представляющая собой древовидную структуру в виде набора связанных узлов (набор каких-либо данных).

Способ представления дерева:

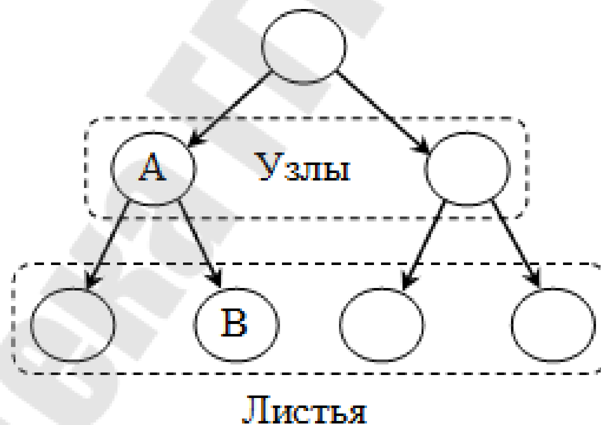

Рис. 3.1. Простое дерево

На самом верхнем уровне иерархии имеется только один узел – корень дерева. Он расположен на уровне с минимальным значением (обычно 0).

Узел *B*, который находится непосредственно под узлом *A*, называется потомком *A*. Если *B* находится на уровне *i*, то *A* – на уровне *i*-1. Узел *B* называется предком *A*.

Библиотека ГГТУ им. П. О. Сухого

Максимальный уровень какого-либо элемента дерева называется его глубиной или высотой. Высота дерева – это наибольшая высота его узлов.

Если элемент не имеет потомков, он называется листом или терминальным узлом дерева. Остальные элементы – внутренние узлы (узлы ветвления).

Число потомков внутреннего узла называется его степенью. Максимальная степень всех узлов есть степень дерева.

Число ветвей, которое нужно пройти от корня к узлу x, называется длиной пути к x. Корень имеет длину пути равную 0; узел на уровне i имеет длину пути равную i.

3.1. Бинарные деревья

Бинарное дерево – это частный случай дерева, в котором каждый узел имеет не более двух потомков. Поскольку потомков у узла может не более двух, то их называют левым и правым.

Бинарное дерево применяется в тех случаях, когда в каждой точке вычислительного процесса должно быть принято одно из двух возможных решений. Поэтому чаще всего данный тип дерева широко используется для поиска.

Бинарное дерево обладает следующим свойством: ключ левого потомка всегда меньше ключа родительского элемента, а ключ правого потомка больше или равен ключу родительского элемента. Ключ – это любое необходимое значение, хранящееся в узле, по которому происходит расположение узла в дереве. Примером ключа может служить значение результатов забега участников соревнования, их возраст и т. д.

Пример бинарного дерева (рис. 3.2):

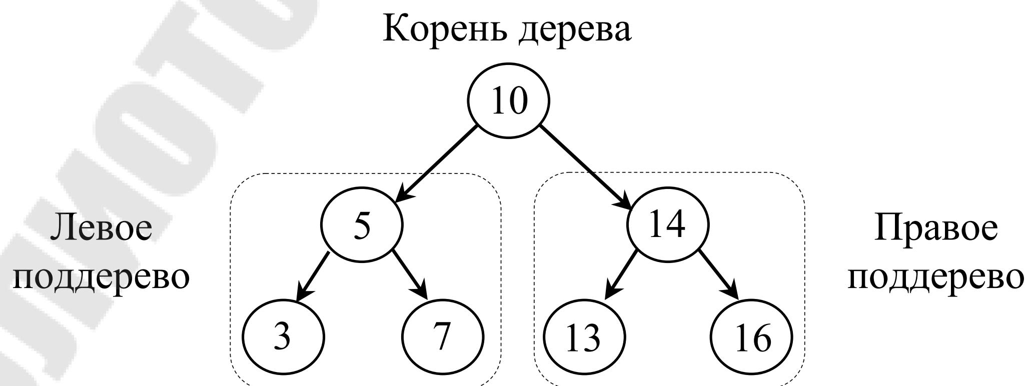

Рис. 3.2. Бинарное дерево

Библиотека ГГТУ им. П. О. Сухого

Высота дерева, как и раньше, определяется количеством уровней, на которых располагаются его узлы.

Если дерево организовано таким образом, что для каждого узла все ключи его левого поддерева меньше ключа этого узла, а все ключи его правого поддерева – больше, оно называется деревом поиска. Одинаковые ключи здесь не допускаются.

Сбалансированными, или AVL-деревьями, называются деревья, для каждого узла которых высоты его поддеревьев различаются не более чем на 1. Причем этот критерий обычно называют AVL-сбалансированностью в отличие от идеальной сбалансированности, когда для каждого узла дерева количество узлов в его левом и правом поддеревьях различаются не более чем на единицу.

Дерево по своей организации является рекурсивной структурой данных, поскольку каждое его поддерево также является деревом. В связи с этим действия с такими структурами чаще всего описываются с помощью рекурсивных алгоритмов.

- Основные действия над бинарными деревьями:
- Построение (создание) дерева.
- Вставка нового элемента.
- Обход дерева.
- Поиск элемента с указанным значением в узле.
- Удаление заданного элемента.
- Удаление дерева.

3.2. Реализация дерева

Узел дерева можно описать как структуру (в качестве ключа используются целые числа):

```
struct treeNode
{
   int value;
   struct treeNode *left;
   struct treeNode *right;
};
```

3.3. Построение дерева

Построение дерева основано на функции вставки нового элемента в дерево, которая будет рассмотрена ниже. Фактически, строя новое дерево, мы поочередно добавляем необходимый элемент в созданное

Библиотека ГГТУ им.П.О.Сухого

дерево. Допустим, мы хотим создать бинарное дерево из элементов: 12, 7, 9, 10, 4, 1, 5, 16, 14, 22, 18 (в таком же порядке). Тогда дерево может выглядеть так, как на рис. 3.3 (в зависимости от алгоритма добавления элемента порядок узлов может быть немного другой):

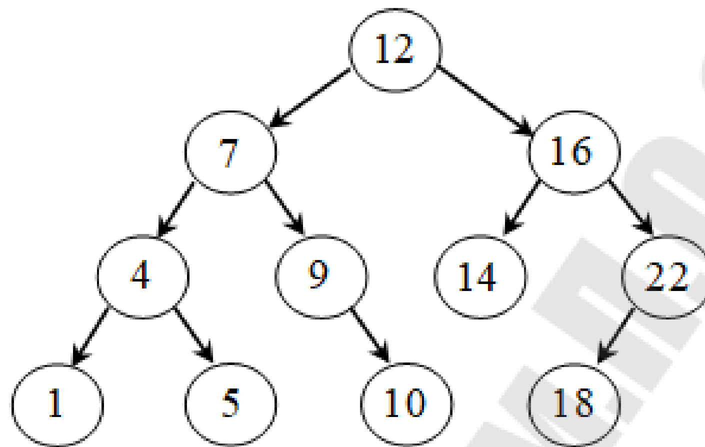

Рис. 3.3. Построение дерева

3.4. Вставка нового элемента в дерево

Для того чтобы произошла вставка нового элемента в дерево, необходимо, чтобы в дереве было найдено место для данного элемента. Для этого используется рекурсивный алгоритм, который, начиная с корня, сравнивает значение ключей в узле с ключом добавляемого элемента. Если ключ добавляемого элемента меньше ключа узла, который сейчас проверяется, то алгоритм идет по левой ветви дерева, иначе – по правой. Когда алгоритм дойдет до узла, из которого не выходит необходимая ветвь для последующего поиска, это означает, что место под новый элемент найдено. Пример реализации данного алгоритма может выглядеть так:

```
 treeNode* addNewNode(int newValue, treeNode *tree)
 {
   if (tree == NULL) {
tree = new treeNode;
     tree->value = newValue;
     tree->left = NULL;
     tree->right = NULL;
   }
   else if (x < tree->value)
```


Библиотека ГГТУ им. П.О. Сухого

```
        tree->left = addNewNode(x, tree->left);
    else
        tree->right = addNewNode(x, tree->right);
    return(tree);
}
```

3.5. Способы обхода дерева

Для просмотра дерева или выполнения какой-либо операции над ним часто используется такое действие как «обход дерева». Схематично изобразим дерево (рис. 3.4):

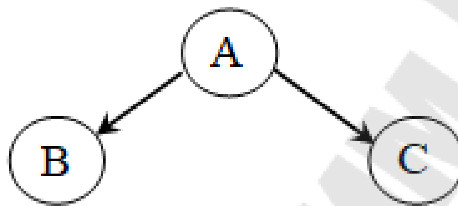

Рис. 3.4. Схема дерева

Существует три способа обхода дерева:

Обход дерева сверху вниз (в прямом порядке): *A*, *B*, *C* – префиксная форма.

Обход дерева в симметричном порядке (слева направо): *B*, *A*, *C* – инфиксная форма.

Обход дерева в обратном порядке (снизу вверх): *B*, *C*, *A* – постфиксная форма.

Тогда пример реализации просмотра дерева в столбец на основе обхода «сверху вниз» может выглядеть следующим образом:

```
void viewTree(treeNode *tree)
{

  if (tree != NULL)
  {
      cout << tree->value;
     viewTree(tree->left);
viewTree(tree->right);
  }
}
```


Библиотека ГГТУ им. П. О. Сухого

На основе этой операции можно организовать функцию вывода в виде дерева:

```
void showTree(Node *&tree, int level)
{
    if (tree)
    {
        showTree(tree->right, level + 1);
        for (int i = 0; i<level; i++)
          cout << "    ";
        cout << tree->value << endl;
        showTree(tree->left, level + 1);
    }
}
```

В таком случае вывод может быть примерно следующим (рис. 3.5):

```
        20:50
    16:40
        14:30
                12:48
                    9:40
            9:30
                8:30
                    7:22
5:36
    4:16
            3:50
        3:45
                2:30
            1:10
```

Рис. 3.5. Вывод в виде дерева

3.6. Поиск элемента в дереве

Операция поиска элементов по ключу в дереве происходит следующим образом: для каждого узла функция сравнивает значение его ключа с искомым ключом. Если ключи одинаковы, то функция возвращает текущий узел, в противном случае функция вызывается рекурсивно для левого или правого поддерева.

```
treeNode* searchNode(int key, treeNode *tree)
{
    if (tree == NULL || key == tree.value)
```


Библиотека ГГТУ им. П.О. Сухого

```
        return x;
    if (key < tree.value)
        return searchNode(key, tree.left);
    else
        return searchNode(key, tree.right);
}
```

3.7. Удаление узла по ключу

При удалении узла из дерева необходимо учитывать, что возможны три ситуации в зависимости от того, сколько потомков имеет удаляемый узел.

1. У удаляемого узла нет потомков, т. е. он является листом. В таком случае у его родителя необходимо просто заменить указатель на удаляемый узел на null. Пример удаления узла со значением ключа = 2 (рис. 3.6):

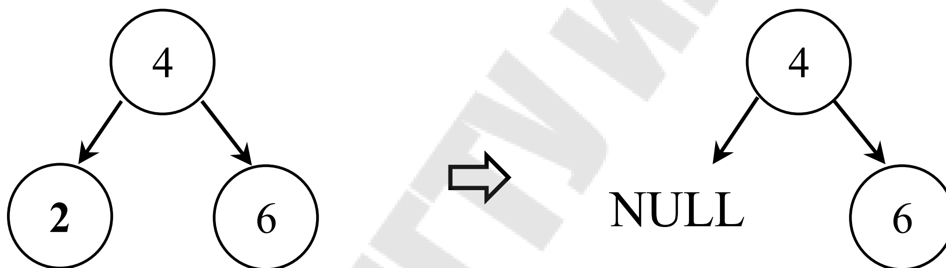

Рис. 3.6. Удаление узла по ключу 2

2. У удаляемого узла есть только один потомок, т. е. из удаляемого узла выходит только одна ветвь. В таком случае происходит соединение родительского и дочернего узлов у удаляемого элемента. Пример удаления узла со значением ключа = 6 (рис. 3.7):

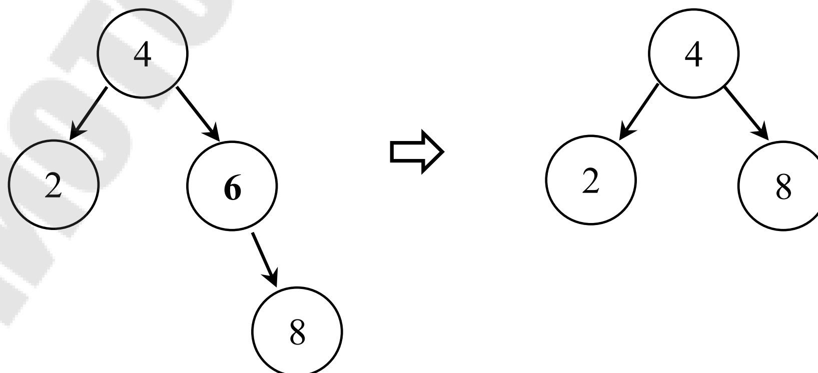

Рис. 3.7. Удаление узла по ключу 6

Библиотека ГГТУ им. П.О. Сухого

3. У удаляемого узла есть два дочерних узла. В таком случае необходимо найти следующий за ним элемент в правом поддереве (у этого элемента не будет левого потомка), либо предыдущий элемент из левого поддерева (у такого элемента не будет правого потомка). В первом случае правого потомка найденного узла подставить на место родителя (на место самого найденного узла), а удаляемый узел заменить найденным узлом. Пример удаления узла со значением ключа = 2 с нахождением следующего за ним элемента (рис. 3.8):

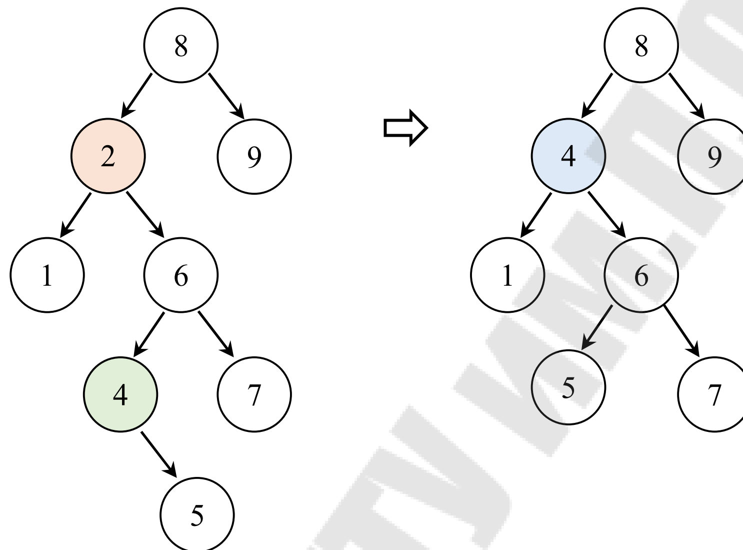

Рис. 3.8. Удаление узла по ключу с нахождением следующего за ним элемента

Пример реализации удаления узла по заданному ключу:

```
treeNode* deleteElement(treeNode *tree, int key)
{
  treeNode *del, *previousForDel, *replaceNode, *previousForReplaceNode;
  del = tree;
  previousForDel = NULL;
  while (del != NULL && del->value != key)
  {
    previousForDel = del;
    if (del->value > key)
      del = del->left;
    else
      del = del->right;
```


Библиотека ГГТУ им.П.О.Сухого

```
  }
  if (del == NULL)
  {
   cout << "Элемент не найден.";
     return tree;
  }
  if (del->right == NULL)
     replaceNode = del->left;
else
     if (del->left == NULL)
        replaceNode = del->right;
   else
   {
          previousForReplaceNode = del;
          replaceNode = del->left;
          while (replaceNode->right != NULL)
           {
          previousForReplaceNode = replaceNode;
          replaceNode = replaceNode->right;
       }
     if (previousForReplaceNode == del)
          replaceNode->right = del->right;
       else
     {
          replaceNode->right = del->right;
          previousForReplaceNode->right = replaceNode->left;
          replaceNode->left = previousForReplaceNode;
       }
  }
  if (del == tree)
     tree = replaceNode;
  else
     if (del->value < previousForDel->value)
        previousForDel->left = replaceNode;
     else
        previousForDel->right = replaceNode;
  cout << "Удален элемент с ключом" << key;
  free(del);
  return tree;
}
```


Библиотека ГГТУ им.П.О.Сухого

3.8. Удаление дерева

Удаление всего дерева (или поддерева – зависит от того, что передадим в функцию) похоже на обход всего дерева и может быть реализовано следующим образом:

```
void freeMemory(treeNode *tree)
 {
   if (tree != NULL) {
      freeMemory(tree->left);
      freeMemory(tree->right);
      delete tree;
   }
}
```

Практическая часть

Разработать проект для обработки дерева поиска, каждый элемент которого содержит структуру данных из варианта курсовой работы. Ключ поиска указан в варианте. В качестве структуры использовать: 1) структуру с указателями на правого и левого потомка; 2) массив с двумя дополнительными полями. Разобраться с балансировками (поворотами) деревьев. В программе должны быть реализованы следующие возможности:

– создание дерева;
– добавление новой записи;
– поиск информации по заданному ключу;
– удаление информации с заданным ключом;
– вывод информации;
– решение индивидуального задания;
– освобождение памяти при выходе из программы.

1. Поменять местами информацию, содержащую максимальный и минимальный ключи.

2. Подсчитать число листьев в дереве.

3. Удалить из дерева ветвь с вершиной, имеющей заданный ключ.

4. Определить глубину дерева.

5. Определить число узлов на каждом уровне дерева.

6. Удалить из левой ветви дерева узел с максимальным значением ключа и все связанные с ним узлы.

Библиотека ГГТУ им. П.О. Сухого

7. Определить количество узлов с четными ключами.

8. Определить число листьев на каждом уровне дерева.

9. Определить число узлов в дереве, имеющих только одного потомка.

10. Определить количество узлов правой ветви дерева.

11. Определить количество записей в дереве, начинающихся с введенной с клавиатуры буквы.

12. Найти среднее значение всех ключей дерева и найти строку, имеющую ближайший к этому значению ключ.

13. Определить количество узлов левой ветви дерева.

14. Определить число узлов в дереве, имеющих двух потомков.

15. Найти запись с ключом, ближайшим к среднему значению между максимальным и минимальным значениями ключей.

16. Подсчитать число узлов в дереве после узла с заданным ключом.

17. Определить количество узлов в дереве, значения которых отличаются от корня не более чем на заданное значение.

18. В левом поддереве вывести значения с четырьмя наименьшими ключами.

19. В правом поддереве вывести значения с тремя наибольшими ключами.

20. В левом поддереве вывести значения с тремя наибольшими ключами.

21. В правом поддереве вывести значения с четырьмя наименьшими ключами.

22. Найти запись с ключом, равным разности между max и min значениями левого поддерева.

23. Найти число узлов, значения которых больше минимального значения и заданного.

24. Найти число узлов, находящихся после узла с заданным значением (листья не считать).

25. Найти среднее значение для всех узлов в дереве, имеющего только одного потомка.

26. Определить три узла, значения которых наиболее близки к корневому.

27. Определить наиболее длинную ветку их узлов, чередующих следующим образом: правый потомок, левый потомок, правый, левый и т. д.

28. Вывести наиболее длинную ветку, содержащую правых потомков.

Библиотека ГГТУ им. П.О. Сухого

29. Вывести наиболее длинную ветку, содержащую левых потомков.

30. Организовать обработку данных типа дерево, в структуре которой имеется указатель на родительский узел.

ГЛАВА 4. ПРЕПРОЦЕССОР

Препроцессор Си/С++ – это программа, которая готовит код программы к этапу компиляции. Препроцессор выполняет макроподстановку, условную компиляцию, включение именованных файлов и другие операции.

Препроцесирование происходит в нескольких логических последовательностях.

• Происходит замена триграфов на соответствующие им символы.

• Выбрасываются пары символов, состоящие из обратной наклонной черты с последующим символом новой строки; тем самым осуществляется «склеивание» строк.

• Выполняются макроподстановки и выполняются директивы препроцессора, а также комментарии заменяются на пробел.

• Замена эслекп последовательностей на соответствующие символы, а соседние строковые литералы по возможности конкатерируются.

• Происходит сборка проекта и подключение всех файлов, ссылки на которые были прописаны в коде программы.

4.1. Трехзнаковые последовательности

Триграф (в семействе языков Си) – последовательность из трех символов, первые два из которых – вопросительные знаки («??»), а третий указывает на значение триграфа.

Появление данных конструкций связано с тем, что некоторые кодировки не поддерживают данных символов, и для того чтобы этот недостаток разрешить, использовались триграфы.

В настоящее время в использовании триграфов нет необходимости.

Список триграфов и их эквивлентов:

```
??=   #
??/   \
??'   ^
??(   [
??)   ]
??!   |
```


Библиотека ГГТУ им. П.О.Сухого

??<   {
??>   }
??-   ~

4.2. Макроопределение и макрорасширение

Макроопределения – это набор пар «идентификатор–лексема (лексемы)», которые объявляются с помощью ключевого слова (директивы препроцессора) #define. Повторение директивы с таким же идентификатором вызовет ошибку.

#define идентификатор(список-идентификаторов) последовательность-лексем

Наравне с #define существует обратная ей директива #undef, которая удаляет ранее определенную пару «идентификатор–значение» директивой #define. Использование для необъявленной пары директивы #undef ошибкой не считается.

Примеры макроопределений:

#define TABSIZE 100
#define ABSDIFF(a, b) ((a)>(b) ? (a)-(b) : (b)-(a))

4.3. Включение файлов

Директива #include предлагает компилятору включить другой исходный файл, имя которого указывается после директивы. При указании имени файла в угловых скобках поиск подключаемого файла будет осуществляться в стандартных каталогах предназначенных для хранения заголовочных файлов стандартной библиотеки языка Си. При использовании кавычек при записи имени файла поиск будет осуществляться в текучем каталоге относительно файла, в котором осуществляется подключение.

include <имя файла>
include “имя файла”

Подключаемые файлы могут содержать в себе подключение других файлов. И также важно следить за тем, чтобы при разработке многофайлового проекта подключаемый файл подключался только единожды в проект.

Библиотека ГГТУ им.П.О.Сухого

4.4. Условная компиляция

Условная компиляция – это возможность программиста управлять процессом компиляции приложения в зависимости от различных факторов, позволяет создавать различные версии приложения из одного кода.

Зачастую такой подход используется для настройки программы под платформу компилятора или возможность управления файлами, которые подключаются единожды для избежания дублирования.

Для предотвращения повторного подключения одного и того же файла используют следующую конструкцию:

```
# ifndef FOO_H
# define FOO_H
...(код заголовочного файла)...
# endif
```

Данная конструкция используется для защиты от повторного подключения одного и того же файла, путем проверки существования данного макроса. Определение макроса происходит при первом подключении файла. При повторном подключении макрос уже будет определен и препроцессор пропустит все, что содержится в этом файле.

Условия препроцессора можно задавать несколькими способами, например:

```
# ifdef x
...
# else
...
# endif
Или
# if x
...
# else
...
# endif
```

Данные конструкции довольно часто используются в заголовочных файлах для различного рода проверок, в том числе и проверку поддержки какой-либо функции, которая может меняться в зависимо-

Библиотека ГГТУ им. П. О. Сухого

сти от платформы. В настоящее время данную конструкцию стараются не использовать.

Также существует управляющая строка вида:

 # pragma последовательность-лексемнеоб

Она призывает препроцессор выполнить зависящие от реализации действия. Неопознанная прагма игнорируется.

4.5. Заранее определенные имена

Существует несколько заранее определенных системных констант, которые содержат служебную информацию. Данные идентификаторы невозможно переопределять или удалять, используя директивы #define и #undef.

Некоторые из идентификаторов:

LINE Номер текущей строки исходного текста, десятичная константа.

FILE Имя компилируемого файла, строка.

DATE Дата компиляции в виде "Ммм дд гггг", строка.

TIME Время компиляции в виде "чч: мм: ее", строка.

ГЛАВА 5. ЯЗЫК С++

С++ – компилируемый, статически типизированный язык программирования общего назначения, совмещающий в себе возможности создания программ как в объектно-ориентированном стиле, так и в процедурном. В язык входит большое количество стандартных библиотек, которые содержат в себе реализацию базовых конструкций и алгоритмов.

Язык С++ отличается в большей степени од своего предшественника Си наличием объектно-ориентированного подхода к созданию приложений. При разработке нового языка С++ одним из подходов было сохранение совместимости с языком Си.

Стоит отметить, что язык программирования С++ не является надмножеством языка программирования Си. И с развитием языка программирования С++ и внесением в него нововведений подход совместимости не всегда был актуален и соблюден. В связи с этим не каждая программа, написанная на языке программирования Си, будет без ошибок скомпилирована компилятором языка С++.

Библиотека ГГТУ им. П.О. Сухого

Язык программирования С++ довольно широко используется в современной разработке программных продуктов в различных сферах. Данный язык подходит для создания компьютерных игр, десктопных приложений, реализации ресурсоемких алгоритмов и пр.

Язык программирования С++, как было указано ранее, является объектно-ориентированным языком.

Объектно-ориентированное программирование – методология программирования, основанная на представлении программы в виде совокупности объектов, каждый из которых является экземпляром определенного класса, а классы образуют иерархию наследования.

В данном определении появилось слово класс и экземпляр класса. Данные определения тоже необходимо ввести для дальнейшего понимания подхода объектно-ориентированного программирования.

Класс – универсальный, комплексный тип данных, состоящий из тематически единого набора «полей» (переменных более элементарных типов) и «методов» (функций для работы с этими полями), т. е. он является моделью информационной сущности с внутренним и внешним интерфейсами для оперирования своим содержимым (значениями полей).

Объект – сущность в адресном пространстве вычислительной системы, появляющаяся при создании экземпляра класса (например, после запуска результатов компиляции и связывания исходного кода на выполнение).

Из определений выше следует, что класс представляет из себя некоторый шаблон объекта, а объект класса – это конкретный объект. Для примера возьмем и опишем с помощью ООП человека. Классом будет является описание, т. е. у человека есть имя, фамилия и прочие параметры, какие-то возможности и функции – способность говорить свое имя, имеет возможность изменить свое имя, фамилию и прочие методы. Данное описание является описанием класса. Когда мы берем и описываем конкретного человека со своим именем, то он является объектом этого класса, другими словами представителем данного класса. Опишем данные конструкции, используя язык программирования С++.

Для объявления нового класса в языке С++ используется ключевое слово ***class***. Создадим класс Human (человек) по нашему словесному описанию выше.

```
class Human{
    private:
```


Библиотека БГТУ им. Сухого

```
		string name;
		string surname;
	public:
		string getFullName(){
			retutn name + " " + surname;
		}
		void setName(string a){
			name = a;
		}
		void setSurname(string a){
			surname = a;
		}
}
```

Приведенный выше пример описания класса человек включает в себя два текстовых поля и три метода класса.

При описании класса также были использованы новые ключевые слова, которые ранее нам не встречались – private, public. Данные слова обозначают модификаторы доступа. С помощью данных конструкций можно управлять доступностью данных и методов вне класса. Модификатор приват обозначает, что данные поля и методы доступны только в рамках данного класса, модификатор паблик говорит, что данные методы и поля доступны в любом месте программы.

Мы создали класс Человек (Human), теперь приступим к созданию объекта класса. Для создания объектов класса используется конструктор. В данном классе мы не описывали никаких конструкторов, что означает – конструктор по умолчанию.

Конструктор – это особая функция-член, инициализирующая экземпляр своего класса. Конструкторы имеют имена, совпадающие с именами классов, и не имеют возвращаемых значений. У конструктора может быть любое число параметров, а у класса – любое число перегруженных конструкторов. Если вы не определили ни одного конструктора, компилятор создаст конструктор по умолчанию, не имеющий параметров.

Наряду с конструктором существует и деструктор. Деструктор – специальный метод класса, служащий для деинициализации объекта. Выполнение деструктора происходит перед удалением объекта. Задается деструктор точно также, как и конструктор, только перед его названием ставится знак тильда (~).

Библиотека ГГТУ им.П.О.Сухого

Для создания объекта класса нужно объявить новую переменную с типом данных нашего нового класса:

```
...
Human  person1 = new Human();
person1.setName("Иван");
persin1.setSurname("Иванов");
person1.getFullName();
```

При объявлении переменной person1 была выполнена конструкция с оператором new. Данная конструкция используется для вызова конструктора класса для инициализации объекта класса. После того как объект был инициализирован, используя переменную person1 для обращения к объекту класса, выполнили методы для задания имени и фамилии, а после получили полное имя человека.

Обращение к методам и полям класса осуществляется через точку, причем обратиться можно только к public полям и методам. И поэтому в нашем примере мы не можем никак обратиться к полям класса кроме как через методы для того, чтобы задать значение имени и фамилии.

5.1. Основные концепции (объектно-ориентированное программирование)

ООП-подход основывается на трех принципах: наследование, инкапсуляция, полиморфизм.

Инкапсуляция — возможность регулирования доступа к различным частям кода. Данный подход позволяет скрывать реализацию для невозможности вмешательства извне, а также предотвратить неправильное использование данных.

Наследование – свойство системы, позволяющее описать новый класс на основе уже существующего с частично или полностью заимствующейся функциональностью. Класс, от которого производится наследование, называется базовым, родительским или суперклассом. Новый класс – потомком, наследником, дочерним или производным классом.

Полиморфизм – свойство системы, позволяющее использовать объекты с одинаковым интерфейсом без информации о типе и внутренней структуре объекта.

Библиотека ГГТУ им. П.О. Сухого

5.2. Инкапсуляция

Инкапсуляция – механизм, с помощью которого возможно управление доступности полей и методов за пределами класса. Это дает нам возможность скрывать реализацию класса от доступа извне. Таким образом поля и методы защищаются от неправильного использования.

В С++ существует несколько модификаторов доступа:

– Public – поля и методы доступны и внутри класса, и за его пределами;

– Private – поля и методы доступны только внутри класса;

– Protected – поля и методы доступны внутри класса и в классах наследниках.

Открытые данные (Public) доступны за пределами объекта и внутри него. Если методы и данные являются открытыми, то, несмотря на то, что они заданы внутри объекта, они доступны и для других частей программы.

Внутри объекта методы и данные могут быть закрытыми (private). Закрытые методы и данные доступны только для других частей этого объекта. Таким образом, закрытые методы и данные недоступны для тех частей программы, которые существуют вне объекта.

Существуют также методы и данные, доступные внутри класса, а также всем классам, которые наследуют этот класс. Такие данные и методы помечаются модификатором доступа Protected.

Для демонстрации приведем код:

```
class Base {
protected:
void protected_f() {}
private:
void private_f() {}
public:
void base_f() {
private_f();
Base b;
b.private_f();
}
};
class Derived : public Base {
void derived_f() {
Derived d;
```


Библиотека ГГТУ им. П.О. Сухого

```
d.protected_f();
Base b;
b.protected_f();
((Derived*)(&b))->protected_f();
}
};
```

Практическая часть

Создать в классе функцию для вывода матрицы, метод для получения размерностей массива.

Создать в классе метод для нахождения наибольшего и наименьшего значения в массиве.

Создать функцию для получения элемента по индексу в массиве.

Создать функцию, которая принимает один объект типа массив, результатом работы является новый массив, который содержит пересечение массива содержащегося в объекте класса и массива из входного объекта (например, исходные массивы: [1]–[4] и [3]–[6], результатом пересечения двух множеств будет [3], [4]).

Создать функцию в соответствии с индивидуальным заданием из списка ниже в соответствии с вашим вариантом.

Для реализации класса задания можно создавать дополнительные функции внутри класса, и все эти функции должны быть скрыты в реализации класса.

В программе создать три объекта класса массив. Первый объект класса инициализировать путем ввода с клавиатуры, второй объект – считать с файла, а третий объект создать из результата пересечения первого и второго объекта класса массив.

Вывести на экран для всех трех объектов: размерность массива, элементы массива, наибольшее и наименьшие значения, результат индивидуального задания.

Функция должна возвращать:

1. Сумму положительных элементов, стоящих после второго максимума.

2. Сумму отрицательных элементов, стоящих после второго нулевого элемента.

3. Сумму отрицательных элементов, стоящих после второго максимума.

4. Сумму положительных и отрицательных элементов, стоящих между максимумом и минимумом.

Библиотека ГГТУ им.П.О.Сухого

5. Количество нулевых элементов, стоящих после максимума.

6. Среднее арифметическое значение всех отрицательных и положительных элементов, стоящих после второго нулевого элемента.

7. Среднее геометрическое значение всех отрицательных и положительных элементов, стоящих после второго нулевого элемента.

8. Произведение всех отрицательных и положительных элементов, стоящих после второго нулевого элемента.

9. Произведение отрицательных элементов, стоящих после второго максимума.

10. Сумма положительных элементов после последнего отрицательного.

11. Произведение чисел в массиве между минимумом и максимумом.

12. Произведение тех чисел, которые составляют наибольшую убывающую последовательность.

13. Количество чисел в наибольшей возрастающей последовательности.

14. Сумму чисел в наибольшей возрастающей последовательности.

15. Количество цифр в максимальном числе.

16. Сумма отрицательных элементов на четных местах после второго 0.

17. Количество нолей в массиве между минимумом и максимумом.

18. Сумма чисел из данного интервала между первым и последним положительным (интервал задать константами в классе самостоятельно).

19. Сумма положительных элементов после последнего отрицательного.

20. Количество чисел из данного интервала между первым и последним положительным.

21. Количество элементов самой длинной последовательности, состоящей из одинаковых элементов.

22. Среднее арифметическое значение элементов самой длинной последовательности, состоящей из одинаковых элементов.

23. Среднее геометрическое элементов самой длинной последовательности, состоящей из одинаковых элементов.

24. Произведение элементов самой длинной последовательности, состоящей из одинаковых элементов.

25. Сумму элементов самой длинной последовательности, состоящей из одинаковых элементов.

Библиотека ГГТУ им. П.О.Сухого

26. Сумму квадратов элементов самой длинной последовательности, состоящей из одинаковых элементов.

27. Количество элементов кратных трем на промежутке между максимумом и минимумом.

28. Сумму элементов кратных трем на промежутке между максимумом и минимумом.

29. Длину самой большой возрастающей последовательности в массиве после второго нуля.

30. Сумму элементов самой большой возрастающей последовательности в массиве после второго нуля.

ГЛАВА 6. НАСЛЕДОВАНИЕ КЛАССОВ

Наследование – это процесс создания новых классов на основе уже созданных с приобретением всех качество класса родителя. Точнее, объект может наследовать основные свойства другого объекта и добавлять к ним черты, характерные только для него. С помощью наследования существует возможность создавать иерархии классов для организации больших объемов данных. Например, квадрат является частным случаем прямоугольника, который является частным случаем параллелограмма, который является четырехугольником, который в свою очередь является геометрической фигурой. В каждом случае цепочки следующий объект содержит какую-то часть информации о текущем объекте. Поэтому без использования наследования пришлось бы в каждом новом классе описывать одно и то же. Однако при использовании наследования можно описать Ч объект и в порождающем объекте наследовать класс от предшественника. При этом в классе наследнике можно реализовать дополнительный функционал у уже описанного в классе родителе. Изменения, произведенные в классе наследнике, никак не повлияют на родительский класс.

Разберем наследование на примере:

Пусть в базе данных вуза должна храниться информация о всех студентах и преподавателях.

Представлять все данные в одном классе не получится, поскольку для преподавателей нам понадобится хранить данные, которые для студента не применимы, и наоборот. Поэтому необходим балловый класс, который содержал бы все общее для студентов и преподавателей. Создадим родительский класс human и два дочерних студент (student) и преподаватель (teacher). Дочерние классы будут наследовать класс человек.

Библиотека ГГТУ им. П.О.Сухого

И студент, и преподаватель имеет имя, фамилию и отчество, и поэтому эти поля мы опишем в классе человек. Также метод для получения ФИО общий для ПРЕП и СТУД, его также опишем в классе человек. Также создадим конструктор с параметрами, в который будем передавать все три параметра. На этом описание составляющих базового класса закончены.

Код класса human:

```
class human {
public:
human(std::string last_name, std::string name, std::string second_name)
{ this->last_name = last_name;
this->name = name;
this->second_name = second_name;
}
std::string get_full_name() {
std::ostringstream full_name;
full_name << this->last_name << " "
<< this->name << " "
<< this->second_name;
return full_name.str();
}
private:
::string name;
std::string last_name;
std::string second_name;
};
```

В классе human мы в открытом доступе оставили конструктор класса и метод для получения имени человека. А сами поля, содержащие имя, фамилию и отчество, скрыли от изменений извне.

Далее приступаем к классу студент. Создадим в классе студент дополнительное закрытое поле, которое будет содержать в себе оценки, получаемые студентом. Также запишем метод для подсчета среднего балла студента. После описания отличительных возможностей класса студент от класса человек необходимо наследовать класс человек классом студент.

Для того чтобы класс студент стал наследником класса человек, необходимо в первой строке после имени класса поставить двоеточие

Библиотека ГГТУ им. П.О.Сухого

и написать модификатор доступа полей базового класса, после через пробел имя родительского класса. В общем виде это выглядит так:

```
class Derived : [virtual] [access-specifier] Base
{
};
```

Модификатор доступа перед родительским классом говорит, какой будет модификатор доступа у всего, что унаследовал класс. При использовании public ничего не изменится. При использовании protected преобразует все public поля и методы базового класса как protected, а использование private наследует как private, вне зависимости от того, как они заданы в базовом классе.

После указания родительского класса наследнику необходимо создать конструктор для класса студент. Конструктор дочернего класса должен вызвать конструктор (один из конструкторов, если их несколько) своего родительского класса. Конструктору дочернего класса необходимо после двоеточия написать имя конструктора, передать в него необходимые параметры, если таковые нужны. Дальше описание конструктора не отличается от обычного конструктора:

```
class Derived : [virtual] [access-specifier] Base
{
Derived([type name-param-1,…]) : Base([value-1,…])
};
```

В нашем примере мы передаем в конструктор класса студент четыре параметра: имя фамилия отчество и оценки, после в конструктор родительского класса передаем значения имя фамилия и отчество. Дальше описываем логику конструктора класса студент:

```
class student : public human {
public:
// Конструктор класса Student
student( std::string last_name, std::string name, std::string second_name,
std::vector<int> scores ) : human( last_name, name, second_name ) {
this->scores = scores;
}
float get_average_score() {
unsigned int count_scores = this->scores.size();
unsigned int sum_scores = 0;
```


Библиотека ГГТУ им.П.О.Сухого

```
float average_score;
for (unsigned int i = 0; i < count_scores; ++i) {
sum_scores += this->scores[i];
}
average_score = (float) sum_scores / (float) count_scores;
return average_score;
}
private:
std::vector<int> scores;
};
```

Мы создали класс студент, теперь аналогично классу студент создадим класс учитель. Отличие есть только в создаваемых методах и дополнительных полях. В классе учитель создадим поле, в которое будет записано количество учебных часов за семестр у преподавателя и метод для получения этого значения:

```
class teacher : public human {
public: teacher(
std::string last_name,
std::string name,
std::string second_name,
unsigned int work_time
                  ) : human( last_name, name, second_name )
{
  this->work_time = work_time;
 }
unsigned int get_work_time()
{  return this->work_time;
 }
private:
unsigned int work_time;
};
```

6.1. Дружественные функции

Дружественные функции – такие функции, которые находятся вне пределов класса, но имеют доступ к private и protected полям и методам класса. Необходимость в таких функциях возникла для возможности описания одной функции для нескольких классов.

Библиотека ГГТУ им. П.О. Сухого

Для того чтобы функция – не часть класса имела доступ к private-полям и методам класса, необходимо в определение класса поместить объявление этой дружественной функции, используя ключевое слово friend. Объявление дружественной функции начинается с ключевого слова friend и должно находиться только в определении класса.

```
void func() {...}
class A {
...
friend void func();
};
```

Пример использования дружественной функции (https://msdn.microsoft.com/ru-ru/library/465sdshe.aspx)

```
#include <iostream>
using namespace std;
class Point
{
friend void ChangePrivate( Point & );
public:
Point( void ) : m_i(0) {}
void PrintPrivate( void ){cout << m_i << endl; }
private:
int m_i;
};
void ChangePrivate ( Point &i ) { i.m_i++; }
int main()
{
Point sPoint;
sPoint.PrintPrivate();
ChangePrivate(sPoint);
sPoint.PrintPrivate();
// Output: 0
1
}
```

6.2. Множественное наследование

На примере мы уже разобрали наследование одного класса другим. В языке С++ существует такое понятие ка множественное наследование. И не сложно догадаться, что оно означает наличие у дочер-

Библиотека ГГТУ им.П.О.Сухого

него класса двух родительских класса. Общая форма множественного наследования выглядит так:

```
class Derived : [virtual] [access-specifier] Base-1, [[virtual] [access-specifier] Base-n , …]
        {
        };
```

Для лучшего понимания разберем на примере.

Создадим классы X и Y и наполним их некоторыми полями и методами. Создадим класс Z и наследуем ему два ранее созданных класса X и Y. Класс Z имеет теперь доступ к полям как класса X, так и класса Y.

```
#include <iostream> using namespace std; class X { protected:
int key; public:
X (int i=0) {cout << "X constructor" << endl; key = i;};
~X() {cout << "X destroyed" << endl; cin.get();}; }; class Y { protected:
int key; public:
Y (int i=0) {cout << "Y constructor" << endl; key = i;};
~Y() {cout << "Y destroyed" << endl; cin.get();}; };
class A : public X, public Y {
int key; public:
A(int i=0) : X(i+1), Y(i+2
{ key = X::key + Y::key; }
int getkey(void) {return(key);} };
```

При работе с данными классами конструкторы родительских классов будут вызываться в соответствии с порядком их записи при объявлении дочернего класса, с деструкторами вызов будет происходить в обратном порядке.

Также стоит отметить, что в родительских классах могут быть поля и методы с одинаковыми названиями. Для указания нужного поля конкретного класса необходимо указать имя класса и после через двойное двоеточие указать имя метода или поля.

Практическая часть

При выполнении данной работы необходимо определить базовый класс и производные от него классы. Предусмотреть передачу аргументов конструкторам базового класса; использование виртуальных и перегруженных функций; обработку исключительных ситуаций.

Библиотека ГГТУ им. П.О. Сухого

В следующих заданиях требуется создать базовый класс (как вариант абстрактный базовый класс) и определить общие метод show и другие, специфические для данного класса. Создать производные классы, в которые добавить свойства и методы. Часть методов переопределить. Создать массив объектов базового класса и заполнить объектами производных классов. Объекты производных классов идентифицировать конструктором по имени или идентификационному номеру. Информацию считать из текстового файла. Вывести на экран этот массив (использовать метод Show()), отсортировав массив по вычисляемому параметру в классе.

1. Создать базовый класс «Транспортное средство» и производные классы «Автомобиль», «Велосипед», «Повозка». Подсчитать время и стоимость перевозки пассажиров и грузов каждым транспортным средством.

2. Создать базовый класс «геометрические фигуры» и производные классы «треугольник», «квадрат», «окружность». Подсчитать площадь всех геометрических фигур.

3. Создать базовый класс «Научный работник» и производные классы «лаборант», «студент», «доцент». Подсчитать месячный доход.

4. Создать базовый класс «самолет» и производные классы «грузовой», «пассажирский», «частный». Подсчитать среднюю стоимость перелета за один километр.

5. Создать базовый класс «магазин» и производные классы «супермаркет», «гипермаркет», «маленький магазин». Подсчитать стоимость месячной аренды торговой площади.

6. Создать базовый класс «медперсонал» и производные классы «врач», «медсестра», «заведующий отделением». Подсчитать месячный доход каждого с учетом платы за оказание платных услуг.

7. Создать базовый класс «помещение» и производные классы «жилой дом», «склад», «магазин». Подсчитать суммарную стоимость обслуживания всех коммуникаций (свет, газ, отопление).

8. Создать базовый класс «объект электронного ресурса» и производные классы «новость», «вопрос», «комментарий». Подсчитать количество просмотревших данный объект.

9. Создать базовый класс «оборудование» и производные классы «конвейер», «токарный станок», «печь». Подсчитать количество выпускаемой продукции или пропускную способность за единицу времени.

10. Создать базовый класс «порт» и производные классы «грузовой», «гражданский», «промышленный» (использующийся для добычи). Подсчитать прибыль каждого из портов.

Библиотека ГГТУ им. П.О. Сухого

11. Создать базовый класс «работник» и производные классы «программист», «бизнес-аналитик», «дизайнер». Подсчитать загруженность каждого на проекте в процентах (массив из объектов и есть команда проекта)

12. Создать базовый класс «учащийся» и производные классы «школьник», «студент», «аспирант». Подсчитать успеваемость.

13. Создать базовый класс «геометрическая фигура» и производные классы «квадрат», «трапеция», «прямоугольник». Подсчитать периметр.

14. Создать базовый класс «геометрическая фигура» и производные классы «куб», «шар», «конус». Подсчитать объем.

15. Создать базовый класс «треугольник» и производные классы «прямоугольный треугольник», «равнобедренный треугольник», «правильный треугольник». Подсчитать площадь.

16. Создать базовый класс «депозит» и производные классы «стандартный», «семейный», «премиум». Подсчитать чистую прибыль от депозита за 10 лет.

17. Создать базовый класс «товар» и производные классы «техника», «продукты», «машина». Подсчитать налоговую нагрузку на каждый товар.

18. Создать базовый класс «платеж» и производные классы «квитанция за квартиру», «штраф», «чек в магазине». Подсчитать сумму выплат.

19. Создать базовый класс «машина» и производные классы «легковая», «грузовая», «легковая премиум класса». Подсчитать месячную стоимость обслуживания.

20. Создать базовый класс «геометрическая фигура» и производные классы «шар», «конус», «цилиндр». Подсчитать площадь поверхности каждой фигуры.

21. Создать базовый класс «человек» и производные классы «рабочий», «начальник цеха», «бухгалтер». Подсчитать месячную зарплату.

22. Создать базовый класс «геометрическая фигура» и производные классы «квадрат», «трапеция», «прямоугольник». Подсчитать площадь.

23. Создать базовый класс «геометрическая фигура» и производные классы «куб», «шар», «конус». Подсчитать площадь поверхности.

24. Создать базовый класс «транспорт» и производные классы «машина», «корабль», «самолет». Подсчитать среднюю стоимость доставки товара за один километр.

Библиотека БГТУ им. Шухова

25. Создать базовый класс «геометрические фигуры» и производные классы «треугольник», «квадрат», «окружность». Подсчитать периметр геометрических фигур.

26. Создать базовый класс «билет на поезд» и производные классы «плацкарт», «купе», «спальный». Подсчитать стоимость проезда за километр пути.

27. Создать базовый класс «геометрическая фигура» и производные классы «ромб», «прямоугольник», «трапеция». Подсчитать радиус описанной окружности, если данную фигуру описать нельзя выводить –1.

28. Создать базовый класс «геометрические фигуры» и производные классы «куб», «конус», «усеченный конус». Подсчитать объем фигуры.

29. Создать базовый класс «геометрические фигуры» и производные классы «куб», «конус», «усеченный конус». Подсчитать площадь поверхности фигуры.

30. Создать базовый класс «геометрические фигуры» и производные классы «куб», «конус», «усеченный конус». Подсчитать объем основания.

ГЛАВА 7. ПРОСТРАНСТВА ИМЕН И ИСКЛЮЧИТЕЛЬНЫЕ СИТУАЦИИ

7.1. Пространство имен

Пространство имен — это декларативная область, в рамках которой определяются различные идентификаторы (имена типов, функций, переменных и т. д.).

Пространства имен используются для логической организации программного в виде отдельных обособленных групп, с помощью которых обычно прибегают во избежание конфликтов идентификаторов.

В приделах объявленного пространства имен все идентификаторы являются доступными без уточнения пространства имен. Для отображения к идентификатору из-за предела пространства имен стоит обращаться к идентификатору с указанием полного имени:

```
Name_namespase::name_identifier
```


Библиотека ГГТУ им. П.О. Сухого

Также для доступа к идентификатору из конкретного пространства существует объявление using. С помощью using можно указать пространство имен для конкретного идентификатора и в дальнейшем использовать его без уточнения пространства имен. Также с помощью using можно задать пространство имен для всех идентификаторов.

В следующем примере показано объявление пространства имен и продемонстрированы три способа доступа к членам пространства имен из кода за его пределами.

```
namespace ContosoData
{
   class ObjectManager
   {
   public:
      void DoSomething() {}
   };
   void Func(ObjectManager) {}
}
ContosoData::ObjectManager mgr;
mgr.DoSomething();
ContosoData::Func(mgr);
```

Чтобы добавить в область видимости один идентификатор, используйте объявление using:

```
using WidgetsUnlimited::ObjectManager;

ObjectManager mgr;
mgr.DoSomething();
```

Чтобы добавить в область видимости все идентификаторы пространства имен, используйте директиву using:

```
using namespace WidgetsUnlimited;
ObjectManager mgr;
mgr.DoSomething();
Func(mgr);
```

Библиотека ГГТУ им. П.О.Сухого

7.2. Директивы using

Директива using позволяет использовать все идентификаторы из пространства имен без его явного указания. Использовать using рекомендуется только в файле реализации (в файле .cpp). При наличии только одного-двух идентификаторов рациональнее будет использовать объявления using, чтобы добавить только эти идентификаторы в пространство имен, а остальные не добавлять. Если локальная переменная имеет такое же имя, как и переменная пространства имен, то переменная пространства имен будет скрытой. Создавать переменную пространства имен с те же именем, что и у глобальной переменной, является ошибкой.

Без особой необходимости не размещайте директивы using в файлах заголовков (*.h), так как любой файл, содержащий этот заголовок, добавит все идентификаторы пространства имен в область видимости, что может вызвать скрытие или конфликты имен, которые очень трудно отлаживать. В файлах заголовков всегда используйте полные имена.

7.3. Объявление пространства имен и их членов

Зачастую пространства имен объявляют в заголовочных файлах. А реализация функций находится в файле реализации, и идентификаторы таких функций в файле реализации указываются полные.

Пример:

```
#pragma once
namespace ContosoDataServer
{
   void Foo();
   int Bar();

}

#include "contosodata.h"
using namespace ContosoDataServer;

void ContosoDataServer::Foo()
{
  Bar();
}

int ContosoDataServer::Bar(){return 0;}
```


Библиотека ГГТУ им. П.О.Сухого

Пространство имен может быть объявлено в нескольких блоках в одном файле и в нескольких файлах. Компилятор соединит вместе все части во время предварительной обработки и полученное в результате пространство имен будет содержать все члены, объявленные во всех частях. Примером этого является пространство имен std, которое объявляется в каждом из файлов заголовка в стандартной библиотеке.

7.4. Глобальное пространство имен

Если идентификатор не объявлен явно в пространстве имен, он неявно считается входящим в глобальное пространство имен. Без особой необходимости старайтесь не делать объявлений в глобальной области, если это возможно. Исключением является точка входа функция main, которая должна быть в глобальном пространстве имен. Чтобы явно указать глобальный идентификатор, используйте оператор разрешения области видимости без имени (::SomeFunction(x);). Это позволит отличать данный идентификатор от любого другого элемента с таким же именем, находящегося в другом пространстве имен. Кроме того, это облегчит понимание кода.

7.5. Обработка ошибок и исключений

С C++ при написании кода хорошим тоном считается обработка исключительных ситуаций, связанных с ошибками времени выполнения и логическими ошибками. Исключения предоставляют способ формальных, четко определенных для кода, который обнаруживает ошибки для передачи информации по стеку вызовов. Ошибки, возникающие в программах, обычно разделяются на две категории: логические ошибки и ошибки времени выполнения.

Отработка исключений в C++ использует конструкцию, состоящую из ключевых слов: try, catch. Блок Try содержит в себе строки кода, в которых может возникнуть исключительная ситуация, а Catch содержит в себе инструкции, которые выполнятся в случае возникновения исключительной ситуации в блоке try.

Общая форма блоков try и catch показана ниже:

```
try {
catch (тип1 аргумент) {
catch (тип2 аргумент) {
```


Библиотека ГГТУ им. П.О. Сухого

```
catch (тип3 аргумент) {
}
...
catch (типN аргумент) {
}
```

Размеры блока try могут изменяться в больших пределах. Например, блок try может содержать несколько инструкций какой-либо функции либо же, напротив, включать в себя весь код функции main(), так что вся программа будет охвачена обработкой исключений.

Когда исключение сгенерировано, оно перехватывается соответствующей инструкцией catch, обрабатывающей это исключение. Одному блоку try может отвечать несколько инструкций catch. Какая именно инструкция catch исполняется, зависит от типа исключения. Это означает, что если тип данных, указанных в инструкции catch, соответствует типу данных исключения, то только эта инструкция catch и будет исполнена. Когда исключение перехвачено, arg получает ее значение. Перехваченным может быть любой тип данных, включая созданные программистом классы. Если никакого исключения не сгенерировано, т. е. никакой ошибки не возникло в блоке try, то инструкции catch выполняться не будут.

Для генерации исключения существует оператор throw.

```
throw исключение;
```

Инструкция throw должна выполняться либо внутри блока try, либо в функции, вызванной из блока try. В записанном выше выражении исключение обозначает сгенерированное значение.

Если генерируется исключение, для которого отсутствует подходящая инструкция catch, может произойти аварийное завершение программы.

```
#include <stdexcept>
#include <limits>
#include <iostream>

using namespace std;
class MyClass
{
```


Библиотека ГГТУ им. П.О.Сухого

```
public:
  void MyFunc(char c)
  {
    if(c < numeric_limits<char>::max())
      throw invalid_argument("MyFunc argument too large.");
    //...
  }
};

int main()
{
  try
  {
    MyFunc(256);
  }
  catch(invalid_argument& e)
  {
    cerr << e.what() << endl;
    return -1;
  }
  return 0;
}
```

ГЛАВА 8. ПОЛИМОРФИЗМ

Целью полиморфизма, применительно к объектно-ориентированному программированию, является использование одного имени для задания общих действий класса. Иначе говоря, полиморфизм представляет собой способность объекта использовать методы производного класса, который не существует на момент создания базового. В C++ полиморфизм обеспечивается за счет использования производных классов и виртуальных функций.

Виртуальная функция – это функция, объявленная с ключевым словом virtual в базовом классе и переопределенная в одном или в нескольких производных классах. Важно отметить, что при использовании виртуального метода определение, какой метод будет выполняться, происходит во время выполнения программы, а не как с перегрузкой методов на этапе компиляции. Определение, какой метод выполнять на момент его вызова, делается основываясь на типе объекта.

Библиотека ГГТУ им.П.О.Сухого

Виртуальные функции объявляются в классе родителе, а переопределение происходит в классах потомках.

Пример:

```
class Base {
public:
	virtual void who() {
	cout << *Base\n";
	}
};
class first_d: public Base {
public:
	void who() {
	cout << "First derivation\n";
	}
};
class seconded: public Base {
public:
void who() {
	cout << "Second derivation\n*";
}
};
```

В примере выше создается родительский класс, в котором присутствует один метод, который мы пометили как виртуальный. После создали два дочерних класса, которые наследуют базовый. В каждом из дочерних методов мы переопределили наш виртуальный метод.

Виртуальные функции используются для описания одинаковых действий, внутренняя реализация которых отличается, например, нахождение площади разных геометрических фигур. И для квадрата, и для трапеции можно найти площадь, но формула нахождения этой площади разная. И когда мы имеем массив разных фигур, то можем просто в цикле вызывать одноименный метод для всех видов фигур, а уже в момент вызова программа сама выберет, какой конкретно метод вызвать для нахождения площади, основываясь на типе объекта.

Наряду с виртуальными функциями существуют абстрактные функции, или чистые виртуальные функции. Абстрактные функции – функции, реализация для которых отсутствует. Класс, содержащий

Библиотека ГГТУ им. П.О.Сухого

абстрактные функции, также принято называть абстрактным. Абстрактная функция ничего не делает, но определяет параметры и возвращаемое значение. Пример объявления чистой виртуальной функции:

virtual void Abstr (void) = 0;

преобразуем наш предыдущий пример. В родительском классе заменим виртуальный метод на чистый виртуальный метод.

```
class Base {
public:
virtual void who() = 0;
        };
class first_d: public Base {
public:
void who() {
cout << "First derivation\n";
}
};
class seconded: public Base {
public:
void who() {
cout << "Second derivation\n*";
}};
```

По сути мы просто удалили тело функции в родительском классе. В остальном код не изменился. Изменились только возможности работы с этими классами. Объект класса, содержащий виртуальные функции, не может быть создан. Он является в некотором роде шаблоном для дочерних классов, но в отличие от интерфейсов может содержать в себе, помимо виртуальных функций, обычные функции.

8.1. Перегрузка функций

С++ позволяет определять несколько функций с одинаковым именем в одной области. Такие функции называются перегруженными. Перегруженные функции позволяют программистам указывать для функций разную семантику в зависимости от типов и числа аргументов.

Библиотека ГГТУ им. П.О. Сухого

Перегружаемые функции имеют одинаковое имя, но разное количество или типы аргументов. Это разновидность статического полиморфизма, при которой вопрос о том, какую из функций вызвать, решается по списку ее аргументов. Этот подход применяется в статически типизированных языках, которые проверяют типы аргументов при вызове функции. Перегруженная функция фактически представляет собой несколько разных функций, и выбор подходящей происходит на этапе компиляции. Перегрузку функций не следует путать с формами полиморфизма, где правильный метод выбирается во время выполнения, например, посредством виртуальных функций, а не статически.

```
main()
{
   cout<<volume(10);
   cout<<volume(2.5,8);
   cout<<volume(100,75,15);
}

// volume of a cube
int volume(int s)
{
   return(s*s*s);
}

// volume of a cylinder
double volume(double r,int h)
{
   return(3.14*r*r*h);
}

// volume of a cuboid
long volume(long l,int b,int h)
{
   return(l*b*h);
}
```


Библиотека ГГТУ им. П.О.Сухого

В приведенном выше примере объем различных компонентов рассчитывается с использованием вызовов разных функций «volume» с аргументами, различающимися по типу данных или их количеству.

8.2. Перегрузка конструктора

Конструкторы, используемые для создания экземпляров объектов, также могут быть перегружены в некоторых объектно-ориентированных языках программирования. Из-за того, что во многих языках название конструктора предопределено именем класса, может показаться, что существует только один конструктор. Всякий раз, когда требуются несколько конструкторов, они реализованы в виде перегруженных функций. Конструктор по умолчанию не принимает параметров, экземпляр объекта — члены с нулевым значением. Например, конструктор по умолчанию для объекта bill в ресторане может установить Tip до 15 %:

```
Bill()
{
   tip = 15.0;
   total = 0.0;
}
```

Недостатком является то, что он делает два шага, чтобы изменить значение созданного Bill объекта. Ниже показано создание и изменение значений в рамках основной программы:

```
Bill cafe;
cafe.tip = 10.00;
cafe.total = 4.00;
```

Через перегрузку конструктора можно было бы передать чаевые в качестве параметров при создании. Пример показывает перегруженный конструктор с двумя параметрами:

```
Bill(double setTip, double setTotal)
{
   tip = setTip;
   total = setTotal;
}
```


Библиотека ГГТУ им. П.О. Сухого

Теперь функция, которая создает новый объект Bill, может передавать два значения в конструктор и устанавливать элементы данных в один шаг. Ниже показано создание и установка значений:

Bill cafe(10.00, 4.00);

Это может быть полезно для повышения эффективности программ и уменьшения размера кода.

ГЛАВА 9. СТАНДАРТНАЯ БИБЛИОТЕКА ШАБЛОНОВ (STL)

Стандартная Библиотека Шаблонов предоставляет набор хорошо сконструированных и согласованно работающих вместе обобщенных компонентов C++. Особая забота была проявлена для обеспечения того, чтобы все шаблонные алгоритмы работали не только со структурами данных в библиотеке, но также и с встроенными структурами данных C++. Например, все алгоритмы работают с обычными указателями. Ортогональный проект библиотеки позволяет программистам использовать библиотечные структуры данных со своими собственными алгоритмами, а библиотечные алгоритмы – со своими собственными структурами данных. Хорошо определенные семантические требования и требования сложности гарантируют, что компонент пользователя будет работать с библиотекой и что он будет работать эффективно. Эта гибкость обеспечивает широкую применимость библиотеки.

Ядро стандартной библиотеки шаблонов составляют три основных элемента: контейнеры, алгоритмы и итераторы. Они работают совместно один с другим, предоставляя тем самым готовые решения различных проблем программирования.

9.1. Контейнеры

Контейнеры – это объекты, содержащие другие объекты. Существует несколько различных типов контейнеров. Например, класс vector определяет динамический массив, класс queue создает двустороннюю очередь, а класс list обеспечивает работу с линейным списком. Эти контейнеры называются последовательными контейнерами, поскольку в терминологии STL последовательность представляет собой линейный список. Помимо базовых контейнеров библиотека STL

Библиотека ГГТУ им.П.О.Сухого

определяет ассоциативные контейнеры, которые позволяют эффективно находить нужные значения на основе заданных ключевых значений (ключей). Например, класс map обеспечивает доступ к значениям с уникальными ключами, т. е. хранит пару «ключ/значение» и предоставляет возможность находить значение по заданному ключу.

Каждый контейнерный класс определяет набор функций, которые можно применять к данному контейнеру. Например, контейнер списка включает функции, предназначенные для выполнения вставки, удаления и объединения элементов. А стек включает функции, которые позволяют помещать значения в стек и извлекать значения из стека.

9.2. Алгоритмы

Алгоритмы действуют в контейнерах. Они включают возможности инициализации, сортировки, поиска и преобразования содержимого контейнеров. Многие алгоритмы работают с группой (или диапазоном) элементов внутри контейнера.

9.3. Итераторы

Итераторы – это объекты, которые в той или иной степени являются указателями. Они позволяют циклически опрашивать содержимое контейнера практически так же, как это делает указатель для циклического опроса элементов массива. Существует пять типов итераторов:

1. Произвольного доступа (random access). Сохраняют и считывают значения; позволяют организовать произвольный доступ к элементам.

2. Двунаправленные (bidirectional). Сохраняют и считывают значения; обеспечивают инкрементно-декрементное перемещение.

3. Однонаправленные (forward). Сохраняют и считывают значения; обеспечивают только инкрементное перемещение.

4. Входные (input). Считывают, но не записывают значений; обеспечивают только инкрементное перемещение.

5. Выходные (output). Записывают, но не считывают значения; обеспечивают только инкрементное перемещение.

В общем случае итератор, который имеет большие возможности доступа, можно использовать вместо итератора с меньшими возможностями. Например, однонаправленным итератором можно заменить входной итератор.

Библиотека ГГТУ им. П. О. Сухого

9.4. Другие элементы библиотеки STL

STL опирается не только на контейнеры, алгоритмы и итераторы, но и на другие стандартные компоненты. Основными из них являются распределители памяти, предикаты, функции сравнения и объекты-функции.

Каждый контейнер имеет свой распределитель памяти (allocator). Распределители управляют выделением памяти при создании нового контейнера. Стандартный распределитель – это объект класса allocator, но при необходимости (в специализированных приложениях) можно определять собственные распределители. В большинстве случаев стандартного распределителя вполне достаточно.

Некоторые алгоритмы и контейнеры используют специальный тип функции, называемый предикатом (predicate). Существует два варианта предикатов: унарный и бинарный. Унарный предикат принимает один аргумент, а бинарный – два. Эти функции возвращают результаты «истина/ложь», но точные условия, которые заставят их вернуть значение истины или лжи, определяются программистом. В остальной части этой главы, когда потребуется унарная функция-предикат, на это будет указано с помощью типа UnPred. При необходимости использования бинарного предиката будет применяться тип BinPred. В бинарном предикате аргументы всегда расположены в порядке первый, второй относительно функции, которая вызывает этот предикат. Как для унарного, так и для бинарного предикатов аргументы будут содержать значения, тип которых совпадает с типом объектов, сохраняемых данным контейнером.

9.5. Строки в С++

Для работы со строками в С++ существует класс string. Он включен в стандартную библиотеку С++. Данный класс содержит в себе необходимые методы для работы со строками, конструкторы и деструкторы. Для использования в своем проекте класса необходимо его подключить в проект с помощью директивы

```
#include <string>
```

Реализация Hello world с использованием класса string:

```
#include <iostream>
#include <string>
```


Библиотека ГГТУ им. П.О. Сухого

```
int main()
{
  std::string str = "Hello, world!";
  std::cout << str << std::endl;
  return 0;
}
```

Для удобной работы со строками в C++ в классе реализован ряд методом, которые содержат алгоритмы, наиболее часто к ним применимые, например копирование части строки в другую или сравнение строк.

Примеры работы со строками в C++.

Инициализация строк при описании и длина строки (не включая завершающий нуль-терминатор):

```
string st( "Моя строка\n" );
cout << "Длина " << st << ": " << st.size()
    << " символов, включая символ новой строки\n";
```

Строка может быть задана и пустой:

```
string st2;
```

Для проверки того, пуста ли строка, можно сравнить ее длину с 0:

```
if ( ! st.size() )
```

или применить метод empty(), возвращающий true для пустой строки и false для непустой:

```
if ( st.empty() )
```

Третья форма создания строки инициализирует объект типа string другим объектом того же типа:

```
string st3( st );
```

Строка st3 инициализируется строкой st. Как мы можем убедиться, что эти строки совпадают? Воспользуемся оператором сравнения (==):

```
if ( st == st3 )
```


Библиотека ГГТУ им. П.О.Сухого

Как скопировать одну строку в другую? С помощью обычной операции присваивания:

```
st2 = st3;
```

Для сцепления строк используется операция сложения (+) или операция сложения с присваиванием (+=). Пусть даны две строки:

```
string s1( "hello, " );
string s2( "world\n" );
```

Мы можем получить третью строку, состоящую из конкатенации первых двух, таким образом:

```
string s3 = s1 + s2;
```

Если же мы хотим добавить s2 в конец s1, мы должны написать:

```
s1 += s2;
```

Операция сложения может сцеплять объекты класса string не только между собой, но и со строками встроенного типа. Можно переписать пример, приведенный выше, так, чтобы специальные символы и знаки препинания представлялись встроенным типом char *, а значимые слова – объектами класса string:

```
const char *pc = ", ";
string s1( "hello" );
string s2( "world" );
string s3 = s1 + pc + s2 + "\n";
cout << endl << s3;
```

Подобные выражения работают потому, что компилятор «знает», как автоматически преобразовывать объекты встроенного типа в объекты класса string. Возможно и простое присваивание встроенной строки объекту string:

```
string s1;
const char *pc = "a character array";
s1 = pc;
```


Библиотека ГГТУ им. П.О. Сухого

9.6. Контейнеры STL

В стандартной библиотеке содержатся набор различных безопасных контейнеров для хранения коллекций объектов. Контейнеры – это шаблоны классов. При объявлении контейнера необходимо указать тип элементов, которые будут храниться в контейнере. Для работы с контейнером существуют заранее определенные методы для добавления удаления элементов, а также другие функции для манипуляции с объектами контейнера.

Для перебора объектов контейнера используются итераторы. Их можно использовать явно с использованием специальных функций, также можно использовать неясный переход с помощью цикла range-for.

Контейнеры можно разделить на три категории: последовательные контейнеры, ассоциативные контейнеры и контейнеры-адаптеры.

9.7. Контейнеры последовательности

Последовательные контейнеры поддерживают указанный пользователем порядок вставляемых элементов.

Контейнер vector ведет себя как массив, но может автоматически увеличиваться по мере необходимости. Он поддерживает прямой доступ и связанное хранение и имеет очень гибкую длину. По этим и многим другим причинам контейнер vector является наиболее предпочтительным последовательным контейнером для большинства областей применения. Если вы сомневаетесь в выборе вида последовательного контейнера, начните с использования вектора. Дополнительные сведения см. в разделе класс vector.

Контейнер array обладает некоторыми преимуществами контейнера vector, однако его длина не обладает такой гибкостью. Дополнительные сведения см. в разделе класс array.

Контейнер deque (двусторонняя очередь) обеспечивает быструю вставку и удаление в начале и в конце контейнера. Он, как и контейнер vector, обладает преимуществами прямого доступа и гибкой длины, но не обеспечивает связанное хранение. Дополнительные сведения см. в разделе класс deque.

Контейнер list – это двунаправленный список, который обеспечивает двунаправленный доступ, быструю вставку и удаления в любом месте контейнера, но не поддерживает прямой доступ к элементам контейнера. Контейнер forward_list – однонаправленный список. Это версия контейнера list только с доступом в прямом направлении.

Библиотека ГГТУ им. П.О. Сухого

9.8. Ассоциативные контейнеры

В ассоциативных контейнерах элементы вставляются в предварительно определенном порядке – например, с сортировкой по возрастанию. Также доступны неупорядоченные ассоциативные контейнеры. Ассоциативные контейнеры можно объединить в два подмножества: сопоставления (set) и наборы (map).

Контейнер map, который иногда называют словарем, состоит из пар «ключ-значение». Ключ используется для упорядочивания последовательности, а значение связано с ключом. Например, map может содержать ключи, представляющие каждое уникальное ключевое слово в тексте, и соответствующие значения, которые обозначают количество повторений каждого слова в тексте.

Map – это неупорядоченная версия unordered_map.

Set – это контейнер уникальных элементов, упорядоченных по возрастанию. Каждое его значение также является и ключом.

Set – это неупорядоченная версия unordered_set.

Контейнеры map и set разрешают вставку только одного экземпляра ключа или элемента. Если необходимо включить несколько экземпляров элемента, следует использовать контейнер multimap или multiset. Неупорядоченные версии этих контейнеров – unordered_multimap и unordered_multiset.

Упорядоченные контейнеры map и set поддерживают двунаправленные итераторы, а их неупорядоченный аналоги — итераторы с перебором в прямом направлении.

9.9. Контейнеры-адаптеры

Контейнер-адаптер – это разновидность последовательного или ассоциативного контейнера, который ограничивает интерфейс для простоты и ясности. Контейнеры-адаптеры не поддерживают итераторы.

Контейнер queue соответствует семантике FIFO (первым поступил – первым обслужен). Первый элемент передается – то есть помещается в очередь – должен быть первым извлечен – то есть удаленных из очереди.

Контейнер priority_queue упорядочен таким образом, что первым в очереди всегда оказывается элемент с наибольшим значением.

Контейнер stack соответствует семантике LIFO (последним поступил – первым обслужен). Последний элемент, отправленный в стек, становится первым извлекаемым элементом.

Поскольку контейнеры-адаптеры не поддерживают итераторы, их нельзя использовать в алгоритмах STL.

Библиотека ГГТУ им. П.О.Сухого

9.10. Класс vector

Класс векторов STL – это класс шаблона контейнеров последовательностей для хранения элементов заданного типа в линейном порядке и быстрого произвольного доступа к любому элементу.

```
template <class Type, class Allocator = allocator<Type>>
class vector
```

Шаблон vector расположен в заголовочном файле <vector>. Данный интерфейс эмулирует работу стандартного массива (например, быстрый произвольный доступ к элементам), а также некоторые дополнительные возможности, вроде автоматического изменения размера вектора при вставке или удалении элементов.

Все элементы вектора должны принадлежать одному типу. Например, нельзя совместно хранить данные типов char и int в одном экземпляре вектора. Класс vector обладает стандартным набором методов для доступа к элементам, добавления и удаления элементов, а также получения количества хранимых элементов.

9.11. Список методов для работы с вектором

Конструкторы:

vector::vector Конструктор по умолчанию. Не принимает аргументов, создает новый экземпляр вектора vector::vector(const vector& c) Конструктор копии по умолчанию. Создает копию вектора c.

Операторы

vector::operator= Копирует значение одного вектора в другой vector::operator== Сравнение двух векторов

Доступ к элементам

vector::at Доступ к элементу с проверкой выхода за границу

vector::operator[] Доступ к определенному элементу

vector::front Доступ к первому элементу

vector::back Доступ к последнему элементу O(1)

Итераторы

vector::begin Возвращает итератор на первый элемент вектора

vector::end Возвращает итератор на место после последнего элемента вектора

vector::rbegin Возвращает reverse_iterator на конец текущего вектора.

Библиотека ГГТУ им. П.О. Сухого

vector::rend Возвращает reverse_iterator на начало вектора.

Работа с размером вектора vector::empty Возвращает true, если вектор пуст

vector::size Возвращает количество элементов в векторе

vector::max_size Возвращает максимально возможное количество элементов в вектора

vector::reserve Устанавливает минимально возможное количество элементов в векторе

vector::capacity Возвращает количество элементов, которое может содержать вектор до того, как ему потребуется выделить больше места.

Модификаторы

vector::clear Удаляет все элементы вектора

vector::insert Вставка элементов в вектор Вставка в конец, при условии, что память не будет перераспределяться

vector::erase Удаляет указанные элементы вектора (один или несколько) O(n)

vector::push_back Вставка элемента в конец вектора

vector::pop_back Удалить последний элемент вектора

vector::resize Изменяет размер вектора на заданную величину

vector::swap Обменять содержимое двух векторов

Другие методы vector::assign Ассоциирует с вектором поданые значения O(n), если установлен нужный размер вектора, O(n*log(n)) при перераспределении памяти

vector::get_allocator Возвращает аллокатор, используемый для выделения памяти

Для демонстрации работы с вектором создадим простую программу, в которой создадим вектор, в котором будет содержаться произвольное количество фамилий студентов, и выведем результат работы на экран.

```
#include <iostream>
#include <vector>
#include <string>

int main()
{
  setlocale(LC_ALL, "");
  std::vector<std::string> students;
```


Библиотека ГГТУ им. П.О.Сухого

```
    std::string buffer = "";

    std::cout << "Вводите фамилии студентов. "
             << "По окончание ввода введите пустую строку" << std::endl;

    do {
       std::getline(std::cin, buffer);
       if (buffer.size() > 0) {
          students.push_back(buffer);
       }
    } while (buffer != "");

    unsigned int vector_size = students.size();

    std::cout << "Ваш вектор." << std::endl;
    for (int i = 0; i < vector_size; i++) {
       std::cout << students[i] << std::endl;
    }
    return 0;
 }
```

9.12. Класс stack

Класс адаптера контейнера шаблона, который предоставляет ограничение функциональности, ограничивая доступ к элементу, который был последним добавлен в некоторый тип базового контейнера. Класс стека используется в том случае, когда важно пояснить, что в контейнере выполняются только операции стека. Принцип работы стека на базовом уровне был описан в п. 2.1.

Для работы со стеком существуют следующие методы:

– функции-члены

(конструктор) Создает stack

(деструктор) Уничтожает stack

operator= Задает значения в адаптере контейнера

– элемент доступа

top Доступ к верхнему элементу

– потенциал

empty Проверяет отсутствие элементов в контейнере, используемом для реализации

Библиотека ГГТУ им. П.О.Сухого

size Возвращает количество элементов в

– модификаторы

push вставляет элемент на верх

pop удаляет верхний элемент

swap Обменивает содержимое

9.13. Класс queue

Класс адаптера контейнера шаблона, предоставляющий ограничения функциональных возможностей для некоторого базового типа контейнера, ограничивая доступ к элементам передней и задней. Элементы могут добавляться на задней панели или удаляться из начала и элементы, которые можно проверить на любом конце очереди:

– функции-члены

(конструктор) Создает queue

(деструктор) Уничтожает queue

operator= Задает значения в адаптере контейнера

– элемент доступа

front Предоставляет доступ к первому элементу

back Предоставляет доступ к последнему элементу

– потенциал

empty Проверяет отсутствие элементов в контейнере, используемом для реализации

size Возвращает количество элементов в контейнере

– модификаторы

push Вставляет элемент в конец

pop удаляет первый элемент

swap Обменивает содержимое

Практическая часть

Написать программу по созданию, просмотру, добавлению и решению поставленной задачи для линейного списка (стек и/или очередь).

1. Создать стек из случайных целых чисел и преобразовать его в два стека. Первый должен содержать только положительные числа, а второй – только отрицательные. Затем удалить элементы по модулю меньшие 5.

2. Для ряда натуральных чисел длиной N>2, представленного в виде стека, построить последовательность:

A1*An, A2*An-1,..., , которую записать в структуру типа очередь.

Библиотека ГГТУ им. П.О. Сухого

3. Описать функцию, которая вставляет в очередь L за первым вхождением элемента E все элементы стека L1, если E входит в L.

4. Из двух стеков сформировать новый следующим образом: первый элемент первого стека, второй – второго, третий - первого и т. д.

5. Из двух стеков сформировать новый следующим образом: сперва записать четные по значению элементы первого стека, затем – четные по значению элементы второго стека.

6. Из стека сформировать новый, состоящий из элементов, перед которыми и после которых, стоят одинаковые значения.

7. Если два стека идентичны, то сформировать очередь из элементов первого стека следующим образом – первый элемент, затем последний; второй, затем предпоследний … .

8. Если стек L1 входит в стек L2, то удалить элементы, входящие в стек L2, равные элементам из L1

9. Составить очередь из N чисел. Проверить ее на наличие одинаковых элементов. Одинаковые элементы перенести в начало очереди.

10. Для стека натуральных чисел длиной N>2 построить новый по следующему принципу:

A1+A3, A2+A4, . . . An-2+An.

11. В стеке переместить минимальный элемент в начало списка, а максимальный – в конец.

12. Из двух стеков сформировать новый следующим образом: сперва записать четные по индексу элементы первого стека, затем – четные по индексу элементы второго стека.

13. Представить автотрассу в виде очереди, элементы которой содержат информацию о названии населенных пунктов и расстоянии между ними. По заданному расстоянию выбрать названия двух населенных пунктов, расстояние между которыми минимально отличается от заданного.

14. Создать стек и разместить в нем в случайном порядке данные о колоде игральных карт. Разработать программу, проводящую перемешивание колоды карт путем сдвига ее частей. Результаты перемешивания вывести на печать (колода – 24 карты).

15. Произвести проверку соблюдения баланса скобок вида '{', '}' в тексте программы. Использовать стек.

16. Составить очередь из N чисел. Проверить ее на наличие одинаковых элементов. Одинаковые элементы удалить и перенести в новую очередь.

Библиотека ГГТУ им.П.О.Сухого

17. Даны очереди *A* и *B*. Удалить из *A* числа, имеющиеся в *B*, добавить в конец очереди *A* числа очереди *B*, которых нет в *A*.

18. Создать очередь из случайных целых чисел. Удалить из этой очереди все элементы, находящиеся между максимальным и минимальным элементами. Сформировать новую очередь из удаляемых элементов.

19. Создать две очереди из случайных целых чисел. Вместо элементов первой очереди, заключенных между максимальным и минимальным элементами, вставить вторую.

20. Создать очередь из случайных положительных и отрицательных целых чисел. Образовать из нее два стека, первый должен содержать значения до первого отрицательного числа, а второй – после первого положительного.

21. Создать две очереди из случайных целых чисел. В первой найти максимальный элемент и за ним вставить элементы второй.

22. Создать очередь из случайных чисел. Определить среднее арифметическое значение всех элементов. Сформировать две новых очереди, поместив в них значения меньшие среднего арифметического и больше соответственно.

23. Даны две очереди, значения в которых отсортированы по возрастанию. Сформировать третью очередь из первых двух, значения в котором также будут отсортированы.

24. Создать две очереди из случайных целых чисел. В первой найти минимальный элемент и удалить элементы, идущие за ним. Во второй найти максимальный элемент и удалить элементы, идущие перед ним.

25. Создать две очереди из целых чисел. Сформировать третью очередь из элементов, которые есть в обоих.

26. Создать три очереди из целых чисел. Сформировать четвертую очередь из элементов, которые есть только в одной из трех очередей, но нет в двух других.

27. Составить очередь из N чисел. Сформировать два новых стека из элементов, стоящих до первого нуля исходного списка и после соответственно.

28. Составить стек из *N* чисел. Удалить из него минимальный элемент. Сформировать новый стек из элементов, которые не равны последнему элементу исходного стека.

29. Составить очередь из *N* чисел. Сформировать новую очередь из элементов исходной очереди, следующих в обратном порядке.

Библиотека БГУИР / www.bsuir.by

30. Составить очередь из *N* чисел. Удалить из очереди элементы, значения которых равны первому элементу очереди. Сформировать новую очередь из элементов, которые не равны минимальному элементу исходной очереди.

ГЛАВА 10. ОСНОВЫ ПРОГРАММИРОВАНИЯ ПОД WINDOWS

Окно (window) – один из наиболее важных объектов в ОС Windows. Окно представляет собой прямоугольную область, в которой приложение отображает выводимую информацию и из которой получает вводимую. Для любого окна Windows определяет дескриптор (handle), который идентифицирует каждое окно.

Класс окна (Window Class) определяет общие свойства и особенности поведения группы окон, созданных на основе класса окна. С помощью класса окна определяются такие характеристики как: пиктограмма, заголовок, кисть для закрашивания, стиль окна и т. д. Одно из полей класса связывает окно с его оконной процедурой, которое обрабатывает сообщения. После прихода сообщения для заданного окна оконная процедура ищет обработчик данного сообщения или использует стандартный Windows-обработчик. Классы окон создаются и регистрируются функциями RegisterClass() и RegisterClassEx(). Создание окон любых видов обеспечивается функциями CreateWindow() и CreateWindowEx().

Оконная процедура (Window Procedure или WndProc) является получа-телем всех сообщений, поступающих окну. Данная процедура имеет спецификатор CALLBACK, который означает, что вызываться данная функция будет напрямую операционной системой. Все сообщения становятся в очередь и выбираются с помощью функции GetMessage. Чаще всего оконная процедура представляет собой конструкцию вида switch–case, распознающую поступившие сообщения и выполняющую их обработку.

Сообщения (Window Message) лежат в основе механизмов событийного управления приложениями. Они представляют собой структуры, передаваемые между приложениями и несущими информацию о событиях, команды, данные и т. д. Для представления сообщений служит структура MSG, поля которой содержат:

typedef struct tagMSG {
HWND hwnd; – дескриптор окна, которому адресовано сообщение

Библиотека ГГТУ им. П.О. Сухого

UINT message; – идентификатор сообщения

WPARAM wparam; – дополнительная информация

LPARAM lparam; – дополнительная информация

DWORD time;– - время отправки сообщения

POINT pt; – экранные координаты курсора мыши в момент отправки сообщения

} MSG;

Минимальное WinAPI-приложение должно содержать как минимум две функции:

WinMain – главную функцию, в которой создается основное окно программы и запускается цикл обработки сообщений;

WndProc – оконную процедуру, обеспечивающую обработку сообщений для основного окна программы.

WinMain является точкой входа в программу и выполняет следующие действия:

– определение класса окна;

– регистрация класса окна;

– создание окна;

– отображение окна;

– запуск цикла обработки сообщений.

Пример такого приложения приведен ниже.

```
#include <windows.h>
#include <tchar.h>
LRESULT CALLBACK WindowProc(HWND, UINT, WPARAM,
LPARAM);
TCHAR szClassWindow[] = TEXT("Каркасное приложение");
INT WINAPI _tWinMain(HINSTANCE hInst, HINSTANCE
hPrevInst,
LPTSTR lpszCmdLine, int nCmdShow)
{
HWND hWnd;
MSG Msg;
WNDCLASSEX wcl;
wcl.cbSize = sizeof (wcl);
wcl.style = CS_HREDRAW | CS_VREDRAW;
wcl.lpfnWndProc = WindowProc;
wcl.cbClsExtra = 0;
wcl.cbWndExtra = 0;
```


Библиотека ГГТУ им. П.О.Сухого

```
wcl.hInstance = hInst;
wcl.hIcon = LoadIcon(NULL, IDI_APPLICATION);
wcl.hCursor = LoadCursor(NULL, IDC_ARROW);
wcl.hbrBackground = (HBRUSH) GetStockOb-
ject(WHITE_BRUSH);
wcl.lpszMenuName = NULL;
wcl.lpszClassName = szClassWindow;
wcl.hIconSm = NULL;

if (!RegisterClassEx(&wcl))
return 0;
hWnd=CreateWindowEx(0,
szClassWindow,
TEXT("Каркас Windows приложения"),
WS_OVERLAPPEDWINDOW,
CW_USEDEFAULT,
CW_USEDEFAULT,
CW_USEDEFAULT,
CW_USEDEFAULT,
NULL,
NULL,
hInst,
NULL);

ShowWindow(hWnd, nCmdShow);

UpdateWindow(hWnd);

while(GetMessage(&Msg, NULL, 0, 0))
{
TranslateMessage(&Msg);
DispatchMessage(&Msg);
}
return Msg.wParam;
}
LRESULT CALLBACK WindowProc(HWND hWnd, UINT uMes-
sage, WPARAM wParam,
LPARAM lParam)
{
```


Библиотека ГГТУ им. П.О. Сухого

```
switch(uMessage)
{
case WM_DESTROY;
PostQuitMessage(0);
break;
default:
return DefWindowProc(hWnd, message, wParam, lParam);
}
return 0;
}
```

ГЛАВА 11. ОБРАБОТКА СОБЫТИЙ

При возникновении любого события операционная система генерирует некоторое сообщение, которое будет обрабатываться оконной процедурой активного окна. Например, когда WinMain вызывает функцию CreateWindow, Windows создает окно и отправляет оконной процедуре сообщение WM_CREATE. Когда WinMain вызывает ShowWindow, Windows отправляет оконной процедуре сообщения WM_SIZE и WM_SHOWWINDOW. Когда WinMain вызывает UpdateWindow, Windows отправляет оконной процедуре сообщение WM_PAINT (перерисовка клиентской части окна).

При завершении работы приложения генерируется сообщение о выходе из программы (WM_QUIT).

При обработке событий клавиатуры генерируются сообщения WM_KEYDOWN (клавиша нажимается) и сообщение WM_KEYUP (клавиша отпускается). В сообщении WM_KEYDOWN содержится также информация о скан-коде нажатой клавиши. Функция API TranslateMessage преобразует пару сообщений WM_KEYDOWN и WM_KEYUP, в символьное сообщение WM_CHAR, которое содержит ASCII-код символа (wParam). Сообщение WM_CHAR помещается в очередь, а на следующей стадии цикла функция GetMessage извлекает его для последующей обработки.

При обработке событий мыши генерируются следующие сообщения:

– WM_MOUSEMOVE – перемещение мыши по клиентской области окна

– WM_LBUTTONDOWN – нажата левая кнопка мыши;

– WM_MBUTTONDOWN – нажата средняя кнопка мыши;

– WM_RBUTTONDOWN – нажата правая кнопка мыши;

Библиотека ГГТУ им.П.О.Сухого

– WM_LBUTTONUP – отпущена левая кнопка мыши;
– WM_MBUTTONUP – отпущена средняя кнопка мыши;
– WM_RBUTTONUP – отпущена правая кнопка мыши.
– WM_LBUTTONDBLCLK – двойной щелчок левой кнопкой мыши;
– WM_MBUTTONDBLCLK – двойной щелчок средней кнопкой мыши;
– WM_MOUSEWHEEL – прокрутка колесика мыши;
– WM_RBUTTONDBLCLK – двойной щелчок правой кнопкой мыши.

Для всех этих сообщений значение параметра lParam содержит положение мыши. При этом в младшем слове (младшие 2 байта) находится значение координаты *X*, а в старшем слове (старшие 2 байта) – значение координаты *Y*. Отсчет координат ведется от левого верхнего угла клиентской области окна. Эти значения можно извлечь из lParam при помощи макросов LOWORD и HIWORD.

Практическая часть

1. Написать приложение, в котором ведется подсчет количества «кликов» левой, правой и средней кнопки мыши. Обновляемую статистику необходимо выводить в заголовок окна.
2. Необходимо при нажатии левой кнопки мыши выводить в заголовок окна сообщение о координатах щелчка, а при нажатии правой кнопки мыши необходимо выводить в заголовок окна информацию о размере клиентской области окна (ширина и высота клиентской области окна).
3. Написать приложение, обладающее следующей функциональностью: при нажатии кнопки <Enter> окно перемещается в левый верхний угол экрана.
4. Написать приложение, обладающее следующей функциональностью: при нажатии кнопок <Shift> <F> окно увеличивается в 2 раза.
5. Написать приложение, обладающее следующей функциональностью: при нажатии кнопок <CTRL> <K> окно смещается влево.
6. Написать приложение, обладающее следующей функциональностью: при нажатии кнопки <стрелка вправо> окно смещается вправо, при нажатии кнопки <стрелка влево> – влево.
7. После 10 щелчков мыши определить – в какой четверти окна произошло больше щелчков. Результат необходимо выводить в заголовок окна.

Библиотека ГГТУ им. П.О. Сухого

8. Написать приложение, обладающее следующей функциональностью: подсчет количеств нажатий клавиш перемещения курсора и вывод информации в заголовок окна.

9. Написать приложение, обладающее следующей функциональностью: при нажатии кнопки <стрелка вверх> окно смещается вверх, при нажатии кнопки <стрелка вниз> – вниз.

10. Написать приложение, обладающее следующей функциональностью: при нажатии клавиши клавиатуры вывести сообщение в заголовок окна о коде нажатой клавиши

11. Написать приложение, обладающее следующей функциональностью: определить за 20 нажатий клавиш клавиатуры – какая клавиша нажималась наибольшее количество раз.

12. Написать приложение, обладающее следующей функциональностью: определить за 10 нажатий кнопок мыши – какая кнопка нажималась наименьшее количество раз.

13. Написать приложение, обладающее следующей функциональностью: определить за 10 нажатий кнопок мыши – самую дальнюю координату щелчка.

14. Написать приложение, обладающее следующей функциональностью: при нажатии левой кнопки мыши – окно смещается влево, при нажатии правой – вправо.

15. Написать приложение, обладающее следующей функциональностью: при нажатии левой кнопки мыши – окно увеличивается, при нажатии правой – уменьшается.

ГЛАВА 12. ГРАФИЧЕСКИЕ ПРИЛОЖЕНИЯ WINDOWS

У программ, написанных для Windows, нет прямого доступа к аппаратной части таких устройств отображения информации, как, например, экран и принтер. Вместо этого они вызывают функции графической подсистемы WinAPI, называемой графическим интерфейсом устройства (Graphics Device Interface, GDI). Функции GDI реализуют основные графические команды при обращении к программным драйверам соответствующих аппаратных устройств. Одна и та же команда (например, LineTo — нарисовать линию) может иметь различную реализацию в разных драйверах. Эта реализация скрыта от программиста, использующего WinAPI, что упрощает разработку приложений.

Библиотека ГГТУ им. П.О. Сухого

GDI (Graphics Device Interface) представляет собой единый унифицированный интерфейс устройств (средств) отображения графической информации в Windows. Работа GDI базируется на понятии контекста устройства (device context – DC), который абстрагирует свойства реальных устройств: экран (окно на экране), принтер, битовый образ в памяти и т. д. Контекст идентифицируется его описателем, тип HDC (handle DC).

Для получения контекста служат функции: GetDC(), GetWindowDC(), GetDCEx(). Они применимы для оконных (экранных) контекстов. Освобождение контекстов выполняется функциями ReleaseDC() для оконных и DeleteDC() для остальных.

Для формирования изображения в контексте служат функции графи-ческих примитивов (например, LineTo(), DrawText() и т. д.) и графические инструменты. Основными объектами являются: «перо» (Pen), «кисть» (Brush) и «шрифт» (Font).

Для создания инструментов служат соответствующие функции GDI API, например CreatePen(), CreateBrush() и так далее. Для сложных объектов может быть определено несколько функций, различающихся параметрами и получаемым эффектом.

Общая схема создания изображения следующая:

– получение контекста;

– установка набора инструментов;

– формирование изображения из примитивов;

– освобождение контекста.

В типичном случае перерисовка содержимого окна инициируется сообщением WM_PAINT.

Установка таймера

Использование таймера является хорошим способом время от времени «будить» программу. Для установки таймера необходимо использовать функцию API SetTimer:

```
UINT SetTimer(
HWND hwnd, UINT nID,
UINT wLength,
TIMEPROC lpTFunc
);
```

Если значение lpTFunc равно NULL, то для обработки сообщений таймера будет вызываться оконная процедура главного окна приложения. В этом случае каждый раз по истечении заданного времен-

Библиотека ГГТУ им. П.О.Сухого

ного интервала в очередь сообщений программы будет помещаться сообщение WM_TIMER, а оконная процедура программы должна будет обрабатывать его так же, как и остальные сообщения. Функция SetTimer в случае успешного завершения возвращает значение идентификатора таймера, в противном случае возвращается 0.

При поступлении сообщения WM_TIMER параметр wParam содержит идентификатор таймера (приложение может установить несколько таймеров), а lParam – адрес функции таймера.

Будучи установленным, таймер будет посылать сообщения до тех пор, пока программа не завершится или не вызовет функцию API KillTimer:

```
BOOL KillTimer(
HWND hwnd,
UINT nID
);
```

Практическая часть

В каждом задании необходимо построить график кусочно-непрерывной функции согласно своему варианту. Установить таймер (10 секунд) для отрисовки осей *X* и *Y*, а также для построения каждого из графиков.

Вариант	Вид функции
1	$y=\begin{cases} 1/x, & \text{если } x \ge -5,\ x \ne 0 \quad (1) \\ x^2, & \text{если } x \le -10 \quad (2) \\ \sqrt{\lvert x+1 \rvert} & \text{в остальных случаях} \quad (3) \end{cases}$
2	$y=\begin{cases} x^2, & \text{если } x \le 0,\ x \ne -10 \quad (1) \\ \sqrt{x+1}, & \text{если } x > 1 \quad (2) \\ 1/x & \text{в остальных случаях} \quad (3) \end{cases}$
3	$y=\begin{cases} x+e^{2x}, & \text{если } x \le 0, \quad x \ne -1 \quad (1) \\ \cos^2 x, & \text{если } 0 < x \le 3{,}14 \quad (2) \\ x & \text{в остальных случаях} \quad (3) \end{cases}$

Библиотека ГГТУ им. П. О. Сухого

Вариант	Вид функции
4	$y=\begin{cases} x^3, & \text{если } x>5, \quad x\neq 20 & (1)\\ x^2, & \text{если } -5\le x\le 5 & (2)\\ \lg\lvert x\rvert & \text{в остальных случаях} & (3)\end{cases}$
5	$y=\begin{cases} \sqrt{x}, & \text{если } x\ge 100, x\neq 105 & (1)\\ \sqrt[3]{x}, & \text{если } x=20 \text{ или } x=40 & (2)\\ x^2+1 & \text{в остальных случаях} & (3)\end{cases}$
6	$y=\begin{cases} \sqrt[3]{x}, & \text{если } x>2 & (1)\\ 1/x, & \text{если } x\le 1 \text{ и } x\neq 0 & (2)\\ x^2-1 & \text{в остальных случаях} & (3)\end{cases}$
7	$y=\begin{cases} 8x+1, & \text{если } x\ge 5, x\neq 9 & (1)\\ x^2+\lvert x\rvert, & \text{если } x\le 1 & (2)\\ x^3+\sqrt{x} & \text{в остальных случаях} & (3)\end{cases}$
8	$y=\begin{cases} 1-3x, & \text{если } x>0, x\neq 8 & (1)\\ x^2-\sin x, & \text{если } x\le -1 & (2)\\ \cos x & \text{в остальных случаях} & (3)\end{cases}$
9	$y=\begin{cases} x^3+1, & \text{если } x\ge 8, x\neq 10 & (1)\\ 2x^2+\sqrt[3]{x}, & \text{если } x\le 1 & (2)\\ \sqrt{x} & \text{в остальных случаях} & (3)\end{cases}$
10	$y=\begin{cases} \sqrt{x}, & \text{если } x\ge 4 \quad \text{и} \quad x\neq 8 & (1)\\ 2x+3, & \text{если } x\le 1 & (2)\\ \lvert x^3-4\rvert & \text{в остальных случаях} & (3)\end{cases}$
11	$y=\begin{cases} \lg^2 2x, & \text{если } x\ge 5 \quad \text{или} \quad x=0{,}5 & (1)\\ 2x^2, & \text{если } x<-2 & (2)\\ \sin x & \text{в остальных случаях} & (3)\end{cases}$
12	$y=\begin{cases} \sqrt{x}, & \text{если } x\ge 4, \text{ или } x=1 & (1)\\ \ln\lvert x+1\rvert, & \text{если } x\le -2 & (2)\\ e^{-x} & \text{в остальных случаях} & (3)\end{cases}$

Библиотека ГГТУ им. П.О.Сухого

Вариант	Вид функции
13	$y=\begin{cases} x/3, & \text{если } -3 \le x \le 3 \quad (1) \\ \lg(x^2+1), & \text{если } x<-3 \quad (2) \\ \sqrt{x^3-2} & \text{в остальных случаях} \quad (3) \end{cases}$
14	$y=\begin{cases} \lvert x^3+4 \rvert, & \text{если } x \le -1 \text{ или } x=0 \quad (1) \\ \sqrt{x/2}, & \text{если } x \ge 8 \quad (2) \\ x^3 & \text{в остальных случаях} \quad (3) \end{cases}$
15	$y=\begin{cases} \sqrt{3x^2+4}, & \text{если } x \ge 2 \quad (1) \\ \ln\lvert x-2 \rvert, & \text{если } x<0 \quad (2) \\ \cos x & \text{в остальных случаях} \quad (3) \end{cases}$
16	$y=\begin{cases} \operatorname{tg} x/2, & \text{если } 0<x \le 2 \quad (1) \\ x^2+1, & \text{если } x \le 0 \quad (2) \\ \cos^2 x & \text{в остальных случаях} \quad (3) \end{cases}$
17	$y=\begin{cases} \sqrt{x^2-2x}, & \text{если } x \ge 10 \quad (1) \\ e^{x/2}, & \text{если } x \le 1 \quad (2) \\ \ln x + x^3/4 & \text{в остальных случаях} \quad (3) \end{cases}$
18	$y=\begin{cases} e^{2x}, & \text{если } x \le 0 \quad (1) \\ \sqrt{\lvert x^2-2 \rvert}, & \text{если } 0<x<7 \quad (2) \\ x/2-x^2 & \text{в остальных случаях} \quad (3) \end{cases}$
19	$y=\begin{cases} \sqrt{e^{2x}}, & \text{если } x \ge 0 \text{ или } x=-1 \quad (1) \\ \cos x/3, & \text{если } x<-1 \quad (2) \\ \lvert x+1 \rvert & \text{в остальных случаях} \quad (3) \end{cases}$
20	$y=\begin{cases} x/3+x^2, & \text{если } 0 \le x \le 3 \quad (1) \\ \lvert x \rvert+3, & \text{если } x<0 \quad (2) \\ \sqrt{2x} & \text{в остальных случаях} \quad (3) \end{cases}$
21	$y=\begin{cases} \sqrt{x+1}, & \text{если } x \ge 8,\ x \ne 10 \quad (1) \\ 0{,}6x, & \text{если } x <= 0 \quad (2) \\ \lg\lvert x \rvert+3 & \text{в остальных случаях} \quad (3) \end{cases}$

Библиотека ГГТУ им. П.О.Сухого

Вариант	Вид функции
22	$y=\begin{cases} 2x^2, & \text{если } x>0, x\neq 3 \quad (1) \\ \sqrt{x^2+1} & \text{если } x\leq -2 \quad (2) \\ \lvert x+5\rvert & \text{в остальных случаях} \quad (3) \end{cases}$
23	$y=\begin{cases} \sqrt{x-1}, & \text{если } x\geq 10, x\neq 20 \quad (1) \\ 1/x+e^{2x}, & \text{если } x<0 \quad (2) \\ \ln(x+1) & \text{в остальных случаях} \quad (3) \end{cases}$
24	$y=\begin{cases} \sqrt{\lvert 2x-x^2-1\rvert}, & \text{если } x\leq -1, x\neq -4 \quad (1) \\ \ln(x+3), & \text{если } x>0 \quad (2) \\ x/2 & \text{в остальных случаях} \quad (3) \end{cases}$
25	$y=\begin{cases} \cos^2 x/2, & \text{если } x>3 \quad (1) \\ \lg\lvert 2x+4\rvert & \text{если } -2{,}5\leq x\leq 3 \quad (2) \\ 3/x & \text{в остальных случаях} \quad (3) \end{cases}$
26	$y=\begin{cases} e^{-x}+1, & \text{если } x\leq 1 \quad (1) \\ \lg 2x, & \text{если } 1<x\leq 5 \quad (2) \\ x^2 & \text{в остальных случаях} \quad (3) \end{cases}$
27	$y=\begin{cases} \lg^2 x/2, & \text{если } x>0, x\neq 2 \quad (1) \\ 2e^{x+1}, & \text{если } x\leq -1 \quad (2) \\ \sqrt{5+x^2} & \text{в остальных случаях} \quad (3) \end{cases}$
28	$y=\begin{cases} x^3/2, & \text{если } x>0, x\neq 2 \quad (1) \\ 2e^{x+1}, & \text{если } x\leq -1 \quad (2) \\ \sqrt{5+x^2} & \text{в остальных случаях} \quad (3) \end{cases}$
29	$y=\begin{cases} \sqrt{x+2}, & \text{если } x\geq 7 \text{ или } x=-1 \quad (1) \\ x-1, & \text{если } 0<x<7 \quad (2) \\ \lvert x^2+4\rvert & \text{в остальных случаях} \quad (3) \end{cases}$
30	$y=\begin{cases} \sqrt{\lg^2 x+1}, & \text{если } x\geq 1 \quad (1) \\ 1/x+e^{x}, & \text{если } x\leq -1 \quad (2) \\ 0{,}5x^2 & \text{в остальных случаях} \quad (3) \end{cases}$

Библиотека ГГТУ им. П.О. Сухого

Учебное электронное издание комбинированного распространения

Учебное издание

ЯЗЫК ПРОГРАММИРОВАНИЯ С/С++

Практикум по дисциплине «Программирование» для студентов специальности 1-40 04 01 «Информатика и технологии программирования» дневной формы обучения

Составитель **Косинов** Геннадий Петрович

Электронный аналог печатного издания

Редактор *Н. В. Гладкова*
Компьютерная верстка *Н. Б. Козловская*

Подписано в печать 19.04.18.
Формат 60x84/16. Бумага офсетная. Гарнитура «Таймс».
Ризография. Усл. печ. л. 4,88. Уч.-изд. л. 4,41.
Изд. № 27.
http://www.gstu.by

Издатель и полиграфическое исполнение
Гомельский государственный
технический университет имени П. О. Сухого.
Свидетельство о гос. регистрации в качестве издателя
печатных изданий за № 1/273 от 04.04.2014 г.
пр. Октября, 48, 246746, г. Гомель

Библиотека ГГТУ им. П.О.Сухого