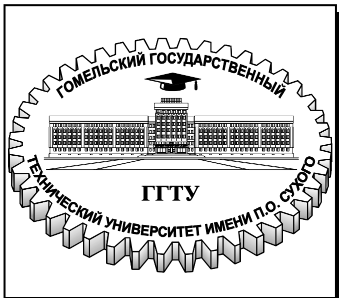


Министерство образования Республики Беларусь

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

Кафедра «Промышленная электроника»

А. В. Сахарук

ТЕХНОЛОГИЯ РАЗРАБОТКИ ПРОГРАММНОГО ОБЕСПЕЧЕНИЯ СИСТЕМ УПРАВЛЕНИЯ: ФУНКЦИИ И КЛАССЫ. ЭЛЕМЕНТЫ ОТОБРАЖЕНИЯ

ПРАКТИКУМ
по выполнению лабораторных работ
по дисциплине «Технология разработки
программного обеспечения систем управления»
для студентов специальности 1-53 01 07
«Информационные технологии и управление
в технических системах»
дневной формы обучения

Электронный аналог печатного издания

Гомель 2018

Библиотека ГГТУ им. П.О. Сухого

УДК 004.43(075.8)
ББК 32.973-018.2я73
С22

Рекомендовано к изданию научно-методическим советом факультета автоматизированных и информационных систем ГГТУ им. П. О. Сухого (*протокол № 5 от 04.12.2017 г.*)

Рецензент: зав. каф. «Информатика» ГГТУ им. П. О. Сухого канд. физ.-мат. наук, доц. *Т. В. Тихоненко*

Сахарук, А. В.

С22 Технология разработки программного обеспечения систем управления: Функции и классы. Элементы отображения : практикум по выполнению лаборатор. работ по дисциплине «Технология разработки программного обеспечения систем управления» для студентов специальности 1-53 01 07 «Информационные технологии и управление в технических системах» днев. формы обучения / А. В. Сахарук. – Гомель : ГГТУ им. П. О. Сухого, 2018. – 62 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 978-985-535-384-4.

Данный практикум продолжает цикл лабораторных работ по дисциплине «Технология разработки программного обеспечения систем управления».

Для студентов специальности 1-53 01 07 «Информационные технологии и управление в технических системах» дневной формы обучения.

УДК 004.43(075.8)
ББК 32.973-018.2я73

ISBN 978-985-535-384-4

© Сахарук А. В., 2018
© Учреждение образования «Гомельский государственный технический университет имени П. О. Сухого», 2018

Библиотека ГГТУ им. П.О. Сухого

Лабораторная работа № 5

Наследование классов, механизм виртуальных функций, переопределение операций

1. Цель работы

Изучить принципы наследования классов, переопределения операций и механизма виртуальных функций.

2. Основные теоретические сведения

2.1. Правила наследования

Наследование является одним из трех основных механизмов ООЯП. В результате использования механизма наследования осуществляется формирование иерархических связей между описываемыми типами. Тип-наследник уточняет базовый тип.

Прототип объявления типа-наследника:

Пример:

```
struct A {    int x,y;
};
struct B: A {    int z;
};
A a1; B b1; b1.x = 1; b1.y = 2; b1.z = 3; a1 = b1;
```

Объект типа *B* наследует свойства объекта типа *A*.

При наследовании наследуются не только информационные члены, но и методы.

Не наследуются:

– Конструкторы
– Деструктор
– Операция присваивания

2.2. Правила видимости при наследовании

Наследование свойств и поведения могут контролироваться с помощью квалификаторов доступа, задаваемых при наследовании: *public*, *protected*, *private*.

Библиотека ГГТУ им. П.О. Сухого

Если не указан тип наследования, то тип наследования по умолчанию определяется описанием типа наследника. Если тип-наследник описывается классом, то тип наследования – закрытый (*private*), если же это структура, то наследование по умолчанию будет открытым (*public*).

Пример:

```
struct A {    int x;
};

class C: A {}; int main(){
        C c;
 // c.x = 1; // ошибка: в классе C из-за закрытого по
// умолчанию наследования поле x
// становится закрытым.
  return 0;
}
```

Если тип-наследник описывается структурой, то наследование по умолчанию становится открытым.

Пример:

```
class A {    public: int x;  private: int y; };

struct C: A {};

int main(){
  C c;
  c.x = 1;     // ошибки нет, т.к. наследование – открытое.
  return 0;
}
```

При необходимости открытого наследования членов базового типа, если тип-наследник описывается с использованием класса, следует явно указывать квалификатор *public*:

```
class C: public A {    int z;
};
```

Библиотека ГГТУ им. П.О. Сухого

Защищенный вид доступа (protected) означает, что члены базового типа в типе-наследнике доступны только для методов своего (базового) типа, а также для методов производного типа. Во всех остальных случаях они ведут себя так же, как члены с закрытым видом доступа (*private*).

2.3. Закрытое (private) наследование

Закрытые члены базового класса не доступны напрямую с использованием дополнительных методов класса-наследника (при любом способе наследования). Работа внутри класса-наследника с такими получаемыми закрытыми членами базового класса возможна только с использованием открытых и защищенных методов базового класса.

При закрытом наследовании открытые и защищенные члены базового класса (любые) доступны только внутри производного класса и недоступны извне (через объекты производного класса), как и его собственные закрытые члены.

2.4. Перекрытие имен

Члены базового класса с именами, совпадающими с именами членов производного класса, доступны в производном классе. Для доступа к ним необходимо указывать квалификатор (имя базового класса) с использованием операции *‘::’*, так как данные члены находятся в доступной области видимости, которая не совпадает с текущей областью видимости. Также метод базового класса доступен через указатель класса-наследника при условии использования квалификатора.

Виртуальную функцию можно использовать, даже если у ее класса нет производных классов. Производный класс, который не нуждается в собственной версии виртуальной функции, не обязан ее реализовывать.

Интерпретация вызова виртуальной функции зависит от типа объекта, для которого она вызывается, в то время как интерпретация вызова не виртуальной функции-члена класса зависит от типа указателя или ссылки, указывающей на этот объект.

Этот механизм делает производные классы и виртуальные функции ключевыми понятиями при разработке программ на С++. Базовый класс определяет интерфейс, для которого производные классы обеспечивают набор реализаций. Указатель на объект класса может

Библиотека ГГТУ им. П.О.Сухого

передаваться в контексте, где известен интерфейс, определенный одним из его базовых классов, но этот производный класс неизвестен.

2.5. Виртуальные функции

Метод класса может содержать спецификатор virtual. Такая функция называется *виртуальной*. Спецификатор virtual может быть использован только в объявлениях нестатических функций-членов класса.

Если некоторый класс содержит *виртуальную* функцию, а *производный* от него класс содержит функцию с тем же именем и типами формальных параметров, то обращение к этой функции для объекта производного класса вызывает функцию, определенную именно в производном классе. Функция, определенная в производном классе, вызывается даже при доступе через указатель или ссылку на базовый класс. В таком случае говорят, что функция производного класса *подменяет* функцию базового класса. Если типы функций различны, то функции считаются разными, и механизм виртуальности не включается.

Порядок выполнения работы

3.1. Простое наследование. Расширение базового класса

Рассмотрим программу. Она содержит в себе три класса:

1) parentClass;
2) childClass;
3) childClass2.

Основной класс parentClass содержит в себе конструктор с двумя входными параметрами целого типа. А также метод summ(), который возвращает сумму входных параметров конструктора. Класс childClass является наследником от parentClass. И в нем добавлен метод proizved (), который возвращает произведение входных параметров конструктора базового класса. Класс childClass2 является наследником обоих предыдущих классов. В нем добавлен метод Raznost (), который возвращает разность входных параметров конструктора первого класса. Рассмотрим листинг программы:

Файл libparentclass.h

```
#ifndef PARENTCLASS_H
#define PARENTCLASS_H
```

Файл libparentclass.cpp

```
#include "libparentclass.h"
parentClass::parentClass() {
```

Библиотека ГГТУ имени П.О.Сухого

```
#include <iostream>
using namespace std;
class parentClass {
public:
   parentClass();
   parentClass(int a_, int b);
   int summ(void);
protected:
   int a;
   int b;
};
#endif // PARENTCLASS_H
```

```
}
parentClass::parentClass(int a_, int b_) {
   a=a_;
   b=b_;
}
int parentClass::summ(void) {
   return a+b;
}
```

Файл libchildclass.h

```
#ifndef CHILDCLASS_H
#define CHILDCLASS_H
#include "libparentclass.h"
class childClass : public parentClass
{
public:
   childClass();
   childClass(int a_, int b_);
   int proizved (void);
};
#endif // CHILDCLASS_H
```

Файл libchildclass.cpp

```
#include "libchildclass.h"
childClass::childClass() {
   cout<<"Constructor child 1"<<endl;
}
childClass::childClass(int a_, int b_) : parentClass(a_,b_) {
   cout<<"Constructor child 2"<<endl;
}
int childClass::proizved() {
   return a*b;
}
```

Файл libchildclass2.h

```
#ifndef CHILDCLASS2_H
#define CHILDCLASS2_H
#include "libchildclass.h"
class childClass2 : public childClass
{
public:
   childClass2(int a_, int b_);
   int Raznost (void);
};
#endif // CHILDCLASS2_H
```

Файл libchildclass2.cpp

```
#include "libchildclass2.h"
childClass2::childClass2(int a_, int b_) : childClass(a_,b_) {
}
int childClass2::Raznost (void) {
   return a-b;
}
```

Библиотека ГГТУ им. П.О.Сухого

Файл main.cpp

```
#include <iostream>
#include "libparentclass.h"
#include "libchildclass.h"
#include "libchildclass2.h"
using namespace std;
int main(int argc, char *argv[]) {
   int a,b;
   cout<<"Enter a=";
   cin>>a;

   cout<<"Enter b=";
   cin>>b;
   parentClass *par = new parentClass(a,b);
   cout<<"Sum = "<<par->summ()<<endl;
   childClass *child = new childClass(a,b);
   cout<<"Sum = "<<child->summ()<<endl;
   cout<<"Proizved = "<<child->proizved()<<endl;
   childClass2 *child2 = new childClass2(a,b);
   cout<<"Sum = "<<child2->summ()<<endl;
   cout<<"Proizved = "<<child2->proizved()<<endl;
   cout<<"Raznost = "<<child2->Raznost()<<endl;
}
```

Примечание. Как уже было сказано, конструкторы не наследуются. Поэтому при необходимости вызова конструктора базового класса при создании наследника необходимо применять конструкцию childClass::childClass(int a_, int b_) : parentClass(a_,b_). Где parentClass(a_,b_) – конструктор базового класса, который необходимо вызвать при вызове конструктора класса наследника childClass(int a_, int b_).

3.2. Виртуальные функции

Как уже было сказано, виртуальные функции предназначены для замены методов базового класса методами класса наследника при вызове наследника через указатель на базовый. Рассмотрим пример использования виртуальной функции при наследовании классов:

Файл libclassvirtual.h

```
#ifndef CLASSVIRTUAL_H
#define CLASSVIRTUAL_H
```

Файл libclassvirtual.cpp

```
#include "libclassvirtual.h"
classVirtual::classVirtual(){
```


Библиотека ГГТУ им.П.О.Сухого

```
#include <iostream>
using namespace std;
class classVirtual {
public:
   classVirtual();
   void f1(void);
   virtual void f2(void);
};

#endif // CLASSVIRTUAL_H
```

```
 }
 void classVirtual::f1(){
    cout<<"parent f1"<<endl;
 }
 void classVirtual::f2() {
    cout<<"parent f2"<<endl;
 }
```

Файл libclassvirtualchild.h

```
#ifndef
CLASSVIRTUALCHILD_H
#define
CLASSVIRTUALCHILD_H
#include "libclassvirtual.h"
class classVirtualChild : public
classVirtual {
public:
   classVirtualChild();
   void f1(void);
   void f2(void);
};
#endif //
CLASSVIRTUALCHILD_H
```

Файл libclassvirtualchild.cpp

```
#include "libclassvirtualchild.h"
classVirtualChild::classVirtualChild()
{
}
void classVirtualChild::f1() {
   cout<<"child f1"<<endl;
}
void classVirtualChild::f2() {
   cout<<"child f2"<<endl;
}
```

Файл main.cpp

```
#include <iostream>
#include "libclassvirtual.h"
#include "libclassvirtualchild.h"
using namespace std;
int main(int argc, char *argv[]) {
   cout << "Start programm" << endl;
   classVirtual *virt = new classVirtual();
   virt->f1();
   virt->f2();
   cout<<"----------------------------------------"<<endl;
   classVirtualChild *virt2 = new classVirtualChild();
```


Библиотека ГГТУ им. П.О. Сухого

```
   virt2->f1();
   virt2->f2();
   cout<<"----------------------------------------"<<endl;
   delete virt;
   virt = new classVirtualChild();
   virt->f1();
   virt->f2();
}
```

3.3. Задание для самостоятельной работы

Необходимо выбрать стандартный базовый класс (например класс string) и реализовать класс-наследник, с двумя оригинальными методами.

4. Содержание отчета

Наименование и цель работы. Краткая основная теоретическая часть, которая применялась вами при выполнении работы. Комментарии, а также снимки экрана для каждого пункта лабораторной работы, которые отражали бы весь ход выполнения. Вывод о полученных навыках и знаниях после выполнения работы.

Контрольные вопросы для самопроверки

1. Что такое наследование?
2. Какие составляющие класса не наследуются?
3. Как влияют квалификаторы доступа на наследование?
4. Что такое виртуальная функция и какова область ее применения?
5. Каким правилам наследования подчиняются конструкторы?
6. Каким правилам наследования подчиняются деструкторы?
7. Каким образом наследуются виртуальные функции?

Библиотека ГГТУ им. П.О. Сухого

Лабораторная работа № 6

Программирование шаблонов функций и классов

1. Цель работы

Изучить принципы создания и использования шаблонов функций и классов.

2. Основные теоретические сведения

2.1. Параметрический полиморфизм

Шаблон представляет собой предварительное описание функции или типа, конкретное представление которых зависит от параметров шаблона. Так, если необходимо написать функции нахождения суммы элементов числовых массивах разных типов (например, *int*, *float* или *double*), то вместо трех различных функций можно написать один шаблон.

2.2. Параметры шаблона

В соответствии со Стандартом ISO 1998 C++ параметром шаблона может быть:

- параметрическое имя типа;
- параметр-шаблон;
- параметр одного из следующих типов:

– интегральный;

– перечислимый;

– указатель на объект любого типа (встроенного или пользовательского) или на функцию;

– ссылка на объект любого типа (встроенного или пользовательского) или на функцию;

– указатель на член класса, в частности, указатель на метод класса.

Интегральные типы:

− знаковые и беззнаковые целые типы;

− bool, char, wchar_t.

При использовании типа в качестве параметра перед параметром, являющимся параметрическим именем типа, необходимо использовать одно из ключевых слов: либо *class*, либо *typename*.

Библиотека ГГТУ им. П.О. Сухого

Таким образом, параметром шаблона не может быть формальный параметр нецелочисленного типа, например, *float* или *double*. Это можно объяснить тем, что основным назначением формальных параметров в качестве параметров шаблона является определение числовых характеристик параметрических типов (например, размер вектора, стека и т. д.).

2.3. Шаблоны функций

Пример шаблона функции:

```
template <class Type>
  Type min2( Type a, Type b ) {
    return a < b ? a : b;
}
```

Обращение:

```
min2( 10, 20 )
```

При обращении к функции-шаблону после имени функции в угловых скобках указываются фактические параметры шаблона – имена реальных типов или значения объектов.

3. Порядок выполнения работы

3.1. Шаблоны функций

Рассмотрим простейший пример использования шаблонов функций. Для этого необходимо создать проект без использования Qt и с компилятором C++. Проект состоит из двух файлов Programm6_1.pro и main.cpp.

Файл Programm6_1.pro

```
TEMPLATE = app
CONFIG += console c++11
CONFIG -= app_bundle
CONFIG -= qt
SOURCES += main.cpp
```

Файл main.cpp

```
#include <iostream>
using namespace std;
template <typename T>
T sum (T a, T b) {
   return a+b;
}
int main() {
   cout << "Template" << endl;
   cout << "sum = " << sum (12,23)
<< endl;
```


Библиотека ГГТУ им. П.О. Сухого

```
    cout << "sum = " << sum
(15.5,13.7) << endl;
    return 0;
}
```

Функция sum возвращает сумму входных параметров. Входные параметры имеют тип T, определенный до нее. При выполнении программы производится два вызова данной функции. В первом случае в качестве входных параметров передается два числа целого типа, а во втором – вещественного. Таким образом функция sum является шаблонной функцией, которая может производить суммирование чисел различных типов.

3.2. Шаблоны классов

Рассмотрим пример простого шаблонного класса. Он представляет собой простейший стек, который позволяет добавлять в него значения, а также извлекать их.

Файл Programm6_2.pro

```
TEMPLATE = app
CONFIG += console c++11
CONFIG -= app_bundle
CONFIG -= qt
SOURCES += main.cpp
HEADERS +=
```

Файл main.cpp

```
#include <iostream>
template <class T> class stack {
private:
    T* body;
    int size;
    int top;
public:
    stack(int sz = 10) {
        size = sz;
        top = 0;
        body = new T[size];
    }
    ~stack() {
        delete[] body;
    }
    T pop() {
        top--;
        return body[top];
    }
    void push(T x) {
        body[top] = x;
```


Библиотека ГГТУ им. П.О. Сухого

```
        top++;
    }
};

using namespace std;
int main() {
    stack<int>  S1( 20);
    stack<char> S2(256);
    stack<double>   S3( 16);
    S1.push(2);
    S1.push(5);
    S1.push(56);
    cout <<  S1.pop()<<" "<<
             S1.pop()<<" "<<
             S1.pop()<<endl;
    S2.push('a');
    S2.push('b');
    S2.push('C');
    cout <<  S2.pop()<<" "<<
             S2.pop()<<" "<<
             S2.pop()<<endl;
    S3.push(3.5);
    S3.push(7.32);
    S3.push(13.99);
    cout <<  S3.pop()<<" "<<
             S3.pop()<<" "<<
             S3.pop()<<endl;
    return 0;
}
```

Класс stack является простейшей реализацией стека. Член класса body является указателем на массив, в котором будут храниться элементы, size – содержит текущий размер стека, а top хранит текущую позицию в стеке. Метод push(T x) добавляет новый элемент в стек, а метод pop() – извлекает текущее значение.

В основной функции программы создается три объекта на основе класса stack: S1, S2 и S3. Объект S1 предназначен для хранения элементов типа int, S2 – элементов типа char, а S3 – элементов типа double.

Библиотека ГГТУ им. П.О.Сухого

Затем в каждый из объектов добавляется по три значения соответствующего типа с помощью метода push(T x) и извлекаются с помощью метода pop().

3.3. Задание для самостоятельной работы

3.3.1. Шаблонные функции

Реализовать шаблонную функцию, которая позволяет вычислять среднее арифметическое для для трех входных параметров.

3.3.2. Шаблонные классы

Реализовать шаблонный класс, который позволит обрабатывать одномерный массив заданного типа. Размер массива должен передаваться в качестве входного параметра конструктора. Добавление нового элемента в массив должно осуществляться с помощью метода add(). Метод at(int i) должен возвращать значение элемента с номером i. Удаление элементов должно производиться с помощью метода remove (int i), где i – номер элемента который должен быть удален. Метод length () должен возвращать текущее количество элементов в шаблоне. Также класс должен содержать методы для обработки массива:

– определение максимального элемента;

– определение минимального элемента;

– подсчет среднего арифметического всех присутствующих элементов массива;

– сортировку элементов массива по возрастанию;

– сортировку массива по убыванию.

При удалении объекта – массив должен освобождать используемую память.

4. Содержание отчета

Наименование и цель работы. Краткая основная теоретическая часть, которая применялась вами при выполнении работы. Комментарии, а также снимки экрана для каждого пункта лабораторной работы, которые отражали бы весь ход выполнения. Вывод о полученных навыках и знаниях после выполнения работы.

Библиотека ГГТУ им. П.О.Сухого

Контрольные вопросы для самопроверки

1. Что такое шаблоны?
2. Для чего предназначены шаблоны?
3. Как описываются шаблонные функции?
4. Как описываются шаблонные классы?
5. Что может быть параметром шаблона по стандарту C++?
6. Что включают в себя интегральные типы?
7. Может ли указатель являться параметром шаблона?

Библиотека ГГТУ им. П. О. Сухого

Лабораторная работа № 7

Файловый ввод-вывод с применением файловых потоков

1. Цель работы

Изучить принципы работы с файлами через потоки.

2. Основные теоретические сведения

Текстовый поток – это последовательность символов. При передаче символов из потока на экран, часть из них не выводится (например, символ возврата каретки, перевода строки).

Двоичный поток – это последовательность байтов, которые однозначно соответствуют тому, что находится на внешнем устройстве.

2.1. Классическая работа с файловыми потоками

В С++ для работы с файловыми потоками используется библиотека fstream.

Для записи информации в файл используется поток вывода ofstream, а для чтения информации из файла применяется поток ввода ifstream.

Для связи файла с потоком вывода необходимо выполнить команду:

ofstream имя логического файла;

Для связи файла с потоком ввода необходимо выполнить команду:

ifstream имя логического файла;

Например:

ofstream OutputFile;

ifstream InputFile;

Для открытия файла применяется метод open соответствующего потока. OutputFile.open (имя физического файла) и InputFile.open (имя физического файла).

Например:

OutputFile.open(“Output.dat”);
InputFile.open(“Input.dat”);

Чтение информации осуществляется аналогично, как и с потоком ввода.

Библиотека ГГТУ им. П.О. Сухого

Например, из файла InputFile нам необходимо прочитать целое число. Для этого выполним следующую команду:

```
int number;
InputFile >> number;
```

Запись информации в файл также аналогична выводу информации в поток вывода. Например, нам необходимо записать целое число 1024 в файл OutputFile. Для этого выполним команду:

```
int number = 1024;
OutputFile << number;
```

Для закрытия необходимо выполнить метод close() файлового потока.

Например закроем файлы InputFile и OutputFile:

```
InputFile.close();
OutputFile.close();
```

2.2. Операции над файлами с использованием стандартных классов Qt

Для работы с файлами в Qt присутствует класс QFile. Для работы с ним необходимо произвести подключение соответствующей библиотеки #include <QFile>.

Рассмотрим основные методы данного класса:

Таблица 7.1

Основные методы класса QFile

Метод	Описание
QFile()	Конструктор умолчания. Не имеет входных параметров. Все необходимые данные устанавливаются с помощью отдельных методов
QFile (const QString &name)	Конструктор, входным параметром является имя открываемого файла. При использовании данного конструктора нет необходимости указывать файл с помощью других методов
copy (const QString &newName)	Метод копирования файлов. Указанный перед вызовом этого метода файл копируется в файл, указанный во входном параметре данного метода. Данный метод возвращает значение типа bool. В случае удачного выполнения операции оно принимает значение true
copy (const QString &fileName, const QString &newName)	Данный метод тоже производит копирование файлов. Однако в отличие от предыдущего имя исходного файла так же, как и имя файла назначения указывается в качестве входного параметра

Библиотека ГГТУ им. П.О. Сухого

Продолжение табл. 7.1

Метод	Описание
exists (const QString &fileName)	Метод проверяет существование файла, указанного во входном параметре. В случае если файл существует – возвращается true
exists() const	Метод проверяет существование ранее указанного файла
fileName() const	Возвращает значение типа QString, в котором указано имя файла, с которым работает данный объект
open (OpenMode mode)	Данный метод осуществляет открытие файла, указанного ранее. В качестве входного параметра передается режим доступа. В случае удачного открытия файла возвращается true
remove()	Метод осуществляет удаление ранее указанного файла. В случае удачи возвращает true
remove (const QString &fileName)	Данный метод аналогичен предыдущему, однако осуществляется удаление файла, указанного в качестве входного параметра
rename(const QString &newName)	Метод производит переименование ранее указанного файла. Новое имя файла принимается в качестве входного параметра. В случае удачи возвращает true
rename (const QString &oldName, const QString &newName)	Метод также производит переименование файла, однако исходное имя файла, как и новое, передается в качестве входного параметра. При успешном выполнении возвращает true
setFileName(const QString &name)	Метод задает имя текущего файла. Данный метод применяется в случае, когда текущий файл не задан в конструкторе либо необходимо изменить текущий файл
read (qint64 maxSize)	Данный метод читает из файла заданное количество байт. Возвращает массив типа QByteArray
readAll ()	Метод читает файл полностью и возвращает данные массивом типа QByteArray
write (const QByteArray &byteArray)	Метод записывает в файл массив типа QByteArray. Возвращает количество записанных байт
atEnd() const	Данный метод сигнализирует о конце файла. Если при операции чтения достигнут конец файла, данный метод возвращает true
size() const	Метод возвращает размер текущего файла

Библиотека ГГТУ им. П.О.Сухого

Для открытия файла применяется метод **open**(**OpenMode** *mode*). Как уже было сказано, в качестве входного параметра в него передается режим доступа. Рассмотрим возможные режимы доступа (табл. 7.2):

Таблица 7.2

Режимы доступа к файлу

Режим	**Значение**
QIODevice::NotOpen	Файл не открыт (это значение не имеет смысла передавать в метод open())
QIODevice::ReadOnly	Открытие файла только для чтения данных
QIODevice::writeOnly	Открытие файла только для записи данных
QIODevice::ReadWrite	Открытие файла для чтения и записи данных
QIODevice::Append	Открытие файла для добавления данных
QIODevice::Unbuffered	Открытие файла для непосредственного доступа к данным, в обход промежуточных буферов чтения и записи
QIODevice::Text	Применяются преобразования символов переноса строки в зависимости от платформы. Для ОС Windows, например – \r\n, а для MacOS X и UNIX – /r
QIODevice::Truncate	Все данные файла, по возможности, должны быть удалены при открытии

Пример открытия файла для чтения:

QFile file(“Input.txt”); //Создаем объект типа QFile и передаем имя файла

file.open(QIODevice::ReadOnly); //Открываем файл для чтения

QByteArray buf;

buf = file.readAll(); //Считываем все содержимое файла в массив

Пример открытия файла для записи:

QFile file(“Output.txt”); //Создаем объект типа QFile и передаем имя файла

file.open(QIODevice::writeOnly); //Открываем файл для записи

QString buf = “Test string”; //Формируем строку для записи

file.write(buf.toLatin1()); //Записываем данные в файл

Примечание. Как видно из табл. 7.1, метод write() принимает в качестве параметра массив типа QByteArray. Для того чтобы записать строку, необходимо выполнить преобразование. Преобразование осуществляет метод toLatin1().

Библиотека ГГТУ им. П.О.Сухого

2.3. Применение классов QTextStream и QDataStream при работе с файлами

Класс QTextStream предназначен для чтения текстовых данных. Числовые данные, передаваемые в поток, автоматически преобразуются в текст. Можно управлять форматом их преобразования, например, метод *QTextStream*::*setRealNumberPrecision*() задает количество знаков после запятой. Следует использовать этот класс для считывания и записи текстовых данных, находящихся в формате Unicode.

Класс QDateStream очень похож на предыдущий, но рассчитан для работы с двоичными данными.

Пример записи данных в текстовый файл через поток:

```
QFile file("file.txt");
QTextStream stream(&file);
file.open(QIODevice::writeOnly); //Открываем файл для записи
QString str = "This is a test";
stream << str.toUpper();
file.close();
```

Более подробную информацию по данным классам можно найти в программе Qt Assistant.

3. Порядок выполнения работы

3.1. Файловый ввод-вывод с использованием стандартной библиотеки языка C

Рассмотрим простейший пример работы с файлом в языке С++. Программа осуществляет запись целого числа в файл и считывание его из файла.

Файл main.cpp

```
#include <iostream>
#include <fstream> //Подключаем библиотеку для работы с файловыми потоками
using namespace std; //Задаем пространство имен std по умолчанию для всего проекта
int main(int argc, char *argv[]) {
   cout<<"Enter number"<<endl;
   int number;
   cin>>number; //Считываем число с клавиатуры
```

Библиотека ГГТУ им. П.О.Сухого

```
ofstream *OutputFile = new ofstream; //Динамически создаем объект на основе класса ofstream для записи информации в файл
OutputFile->open("information.dat"); //Открываем файл information.dat, который будет располагаться в папке с исполняемым файлом
*(OutputFile) << number; //Записываем введенное число в файл
OutputFile->close(); //Закрываем файл
delete OutputFile; //Удаляем объект
ifstream InputFile; //Статически объявляем объект на основе класса ifstream  для чтения информации из файла
InputFile.open("information.dat"); //Открываем файл information.dat, находящийся в папке с исполняемым файлом
int inputNumber; //Объявляем переменную для считывания в нее информации из файла
InputFile >> inputNumber; //Считываем информацию
InputFile.close(); //Закрываем файл
cout<<"inputNumber="<<inputNumber<<endl; //Выводим значение переменной на экран
return 0;
}
```

В результате выполнения программы в папке с исполняемым файлом должен появиться файл information.dat, в котором будет записано число, введенное с клавиатуры.

3.2. Ввод-вывод с использованием класса QFile

Рассмотрим консольное приложение Qt, которое выполняет запись и чтение файла через класс QFile.

Для создания консольного приложения с использованием библиотеки Qt необходимо выбрать пункт «Консольное приложение Qt». На рис. 7.1 изображен внешний вид диалогового окна создания проекта.

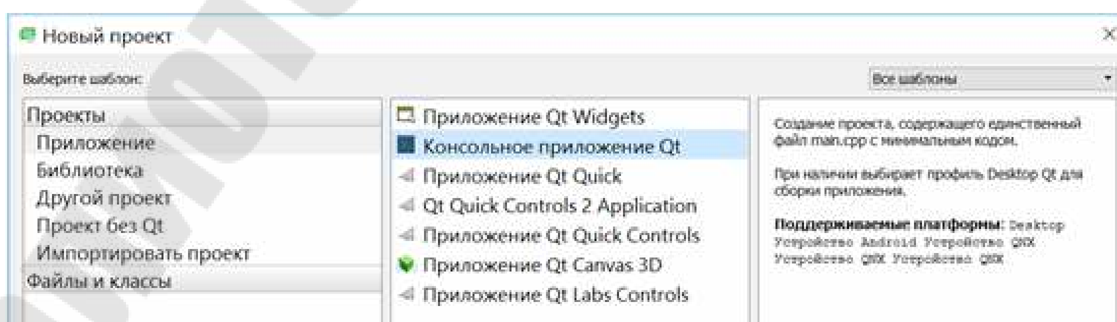


Рис. 7.1. Создание консольного приложения с использованием библиотеки Qt

Библиотека ГГТУ им. П.О.Сухого

Содержание файлов проекта:

Файл Programm7_2.pro
```
QT += core
QT -= gui
CONFIG += c++11
TARGET = Programm7_2
CONFIG += console
CONFIG -= app_bundle
TEMPLATE = app
SOURCES += main.cpp
```

Файл main.cpp
```
#include <QDebug>
#include <QFile>
#include <QByteArray>

int main(int argc, char *argv[]) {
	qDebug()<<"Start programm";
	QString testString = "Test string"; //Создаем тестовую строку
	qDebug()<<"Write to file string: "<<testString; //Выводим ее значение
	экран
	QFile outputFile("information.txt"); //Создаем объект для работы с
	файлом и задаем имя файла
	outputFile.open(QIODevice::WriteOnly); //Открываем файл с оп-
цией доступа по записи
	outputFile.write(testString.toLatin1()); //Записываем строку, пред-
варительно сконвертировав ее в формат QByteArray
	outputFile.close(); //Закрываем файл
	qDebug()<<"File writed"; //Выводим сообщение на экран о том,
что файл записан
	QFile *inputFile = new QFile; //Динамически создаем объект для
работы с файлом
	inputFile->setFileName("information.txt"); //Устанавливаем имя
файла
	inputFile->open(QIODevice::ReadOnly); //Открываем файл для
чтения
	QByteArray readString; //Создаем переменную для хранения счи-
танных данных
```


Библиотека ГГТУ им.П.О.Сухого

```
    readString = inputFile->readAll(); //Считываем весь файл
    inputFile->close(); //Закрываем файл
    delete inputFile; //Удаляем объект
    qDebug()<<"String from file: "<<QString(readString); //Выводим значение считанных данных, сконвертировав их в формат QString
    return 0;
}
```

В результате выполнения программы будет создан файл information.txt, в котором будет содержаться тестовая строка. Файл будет находиться в той же папке, что и исполняемый файл.

3.3. Ввод-вывод с использованием потоков библиотеки Qt

Рассмотрим проект, осуществляющий работу с файлами через потоки библиотеки Qt.

Файл Programm7_3.pro

```
QT += core
QT -= gui
CONFIG += c++11
TARGET = Programm7_3
CONFIG += console
CONFIG -= app_bundle
TEMPLATE = app
SOURCES += main.cpp
```

Файл main.cpp

```
#include <QFile>
#include <QTextStream>
#include <QDebug>
int main(int argc, char *argv[]) {
   qDebug()<<"Start programm";
   QFile *file = new QFile("file1.txt");
   file->open(QIODevice::WriteOnly);
   QTextStream *textStream = new QTextStream(file);
   qDebug()<<"Write information to the file";
   QString testString = "Information for writing to a file";
   *(textStream)<<testString;
   file->close();
   qDebug()<<"Information is writed";

   delete textStream;
   delete file;
   qDebug()<<"Read information";
```


Библиотека ГГТУ им. П.О. Сухого

```
    file - new QFile("file1.txt");
    file-
>open(QIODevice::ReadOnly);
    textStream          =          new
QTextStream(file);
    QString readString;
    readString      =      textStream-
>readAll();
    qDebug()<<"Read    the    string:
"<<readString;
    file->close();
    delete textStream;
    delete file;
    return 0;
}
```

3.4. Задание для самостоятельной работы

Файл содержит несколько строк, в каждой из которых записано единственное выражение вида a#b (без ошибок), где a, b – целочисленные величины, # – операция +, –, /, *. Написать программу, которая считывает эти строки, выводит выражения на экран, а также производит их вычисления. Реализовать два варианта программы:

– работу с файлами реализовать через потоки стандартного С++;

– работу с файлами реализовать с использованием классов Qt.

4. Содержание отчета

Наименование и цель работы. Краткая основная теоретическая часть, которая применялась вами при выполнении работы. Комментарии, а также снимки экрана для каждого пункта лабораторной работы, которые отражали бы весь ход выполнения. Вывод о полученных навыках и знаниях после выполнения работы.

Контрольные вопросы для самопроверки

1. Что такое файл?
2. Что такое текстовый поток?
3. Что такое двоичный поток?

Библиотека ГГТУ им. П.О. Сухого

4. Как организовывается работа с файлами стандартными средствами C++?

5. Назначение класса QFile.

6. Каким образом реализуется работа с файлами через потоки Qt?

7. Какой режим доступа при открытии файла позволяет только читать содержимое?

Библиотека ГГТУ им.П.О.Сухого

Лабораторная работа № 8

Функции обработки исключительных ситуаций

1. Цель работы

Изучить принципы определения и обработки исключительных ситуаций.

2. Основные теоретические сведения

Исключение – это аномальное поведение во время выполнения, которое программа может обнаружить, например: деление на 0, выход за границы массива или истощение свободной памяти. Такие исключения нарушают нормальный ход работы программы, и на них нужно немедленно отреагировать.

Фундаментальная идея обработки исключительных ситуаций состоит в том, что функция, обнаружившая проблему, но не знающая как ее решить, генерирует исключение в надежде, что вызвавшая ее (непосредственно или косвенно) функция сможет решить возникшую проблему. Функция, которая может решать проблемы данного типа, указывает, что она перехватывает такие исключения.

Для реализации обработки исключений в C++ используйте выражения try, throw и catch.

Блок try {...} позволяет включить один или несколько операторов, которые могут создавать исключение.

Выражение throw используется только в программных исключениях и означает, что исключительное условие произошло в блоке try. В качестве операнда выражения throw можно использовать объект любого типа. Обычно этот объект используется для передачи информации об ошибке.

Для обработки исключений, которые могут быть созданы, необходимо реализовать один или несколько блоков catch сразу после блока try. Каждый блок catch указывает тип исключения, которое он может обрабатывать.

Сразу за блоком try находится защищенный раздел кода. Выражение throw вызывает исключение, т. е. создает его.

Блок кода после catch является обработчиком исключения. Он перехватывает исключение, вызываемое, если типы в выражениях

Библиотека ГГТУ им. П.О.Сухого

throw и catch совместимы. Если оператор catch задает многоточие (...) вместо типа, блок catch обрабатывает все типы исключений. Поскольку блоки catch обрабатываются в порядке программы для поиска подходящего типа, обработчик с многоточием должен быть последним обработчиком для соответствующего блока try. Как правило, блок catch(...) используется для ведения журнала ошибок и выполнения специальной очистки перед остановкой выполнения программы.

```
try { ... // защищенный раздел кода
  throw параметр;
}
catch (параметр) { // обработка исключения }
catch (...) { // обработка остальных исключений }
```

3. Порядок выполнения работы

3.1. Простейший пример генерации исключительной ситуации и ее обработки

Рассмотрим простейший пример работы с исключительными ситуациями.

Файл Programm7_1.pro

```
TEMPLATE = app
CONFIG += console c++11
CONFIG -= app_bundle
CONFIG -= qt
SOURCES += main.cpp
```

Файл main.cpp

```
#include <iostream>
using namespace std;
int main() {
  try {
    cout << "Exception: "<<endl;
    throw 1;
    cout << "No exception!"<<endl;
  } catch (int a) {
    cout << "catch:" << a <<endl;
  }
return 0;
}
```

В данной программе часть кода заключена в try-блок. Данный код сначала осуществляет вывод на экран сообщения "Exception: ", затем генерирует исключение с кодом 1, а после выводит сообщение "No exception!". После try-блока расположен обработчик исключи-

Библиотека ГГТУ им. П.О. Сухого

тельных ситуаций с целочисленным кодом catch (int a). При возникновении такой исключительной ситуации на экран будет выведено сообщение "catch:" с числовым кодом данного исключения.

3.2. Применение обработки исключительных ситуаций при использовании функций

В данном примере будет рассмотрено использование исключительных ситуаций при применении функций.

Файл Programm7_2.pro

```
TEMPLATE = app
CONFIG += console c++11
CONFIG -= app_bundle
CONFIG -= qt
SOURCES += main.cpp
```

Файл main.cpp

```
#include <iostream>
using namespace std;
float divAB (int a,int b) {
   float Output;
   if (b == 0) throw -1;
      else Output = a/b;
   return Output;
}
int main() {
  cout<<"Start programm"<<endl;
  int a = 10, b =0;
  try   {
      cout << divAB(a,b) << endl;
  }
  catch (int i) {
     cout<<"Error"<<i<<endl;
  }
  cout<<"End programm"<<endl;
  return 0;
}
```

Функция divAB() осуществляет деление входного параметра a на входной параметр b. В случае если значение b равно 0, генерируется исключение с кодом -1. В основной программе вызов данной функции взять в блок try. В случае генерации исключения с целочисленным кодом вызывается обработчик, который в свою очередь выводит код данной ошибки на экран. Обратите внимание, что генерация исключительной ситуации происходит внутри тела функции, а его обработка – в основной программе.

Библиотека ГГТУ им. П.О. Сухого

3.3. Применение обработки исключительных ситуаций в классах

Рассмотрим пример реализации обработки исключительных ситуаций на примере класса стека.

Файл Programm7_3.pro

```
TEMPLATE = app
CONFIG += console c++11
CONFIG -= app_bundle
CONFIG -= qt
SOURCES += main.cpp \
   stack.cpp
HEADERS += \
   stack.h
```

Файл stack.h

```
#ifndef STACK_H
#define STACK_H
class stack {
private:
   int* body;
   int size;
   int top;
public:
   stack(int sz = 10);
   ~stack();
   int pop();
   void push(int x);
};
#endif // STACK_H
```

Файл stack.cpp

```
#include "stack.h"
stack::stack(int sz) {
   size = sz;
   top = 0;
   body = new int[size];
}
stack::~stack() {
   delete[] body;
}
int stack::pop() {
   top--;
   if (top == 0) throw -1;
   else return body[top];
}
void stack::push(int x) {
   if (top >= size) throw -2;
   else {
```

Файл main.cpp

```
#include <iostream>
#include "stack.h"
using namespace std;
int main() {
   cout << "Start programm" <<
endl;
   stack Stack(10);

   try {
      cout << "Filling the stack" <<
endl;
      for (int i=0;i<10;i++)
         Stack.push(i);
      cout << "Extracting the stack"
<< endl;
      for (int i=0;i<12;i++)
         cout << Stack.pop() << endl;
```

Библиотека ГГТУ им. П.О.Сухого

```
        body[top] = x;
        top++;
    }
}
```

```
    }
    catch (int i) {
        switch (i) {
        case -1:   cout << "Stack is empty" << endl;
            break;
        case -2:  cout << "Stack is full" << endl;
            break;
        }
    }
    return 0; }
```

В данном примере реализован класс стека, у которого производится контроль при добавлении значения и извлечения. Если стек заполнен и производится попытка добавления нового значения, то генерируется исключительная ситуация с кодом -2, а в случае если производится извлечение значения из стека, но он пуст, то генерируется исключительная ситуация с кодом -1.

В основной программе вызовы методов push и pop взяты в try-блок для перехвата исключительных ситуаций. В случае если возникла исключительная ситуация, вызывается catch-обработчик и по коду определяется тип исключительной ситуации.

3.4. Задание для самостоятельной работы

Реализовать класс, который позволит обрабатывать одномерный массив целого типа. Размер массива должен передаваться в качестве входного параметра конструктора. Добавление нового элемента в массив должно осуществляться с помощью метода add(). Метод at(int i) должен возвращать значение элемента с номером i. Удаление элементов должно производиться с помощью метода remove (int i), где *i* – номер элемента, который должен быть удален. Метод length () должен возвращать текущее количество элементов в шаблоне. Также класс должен содержать методы для обработки массива:

– определение максимального элемента;

– определение минимального элемента;

– подсчет среднего арифметического всех присутствующих элементов массива;

– сортировку элементов массива по возрастанию;

– сортировку массива по убыванию.

Библиотека ГГТУ им. П.О.Сухого

При удалении объекта – массив должен освобождать используемую память.

Также продумать возможные исключительные ситуации и реализовать их обработку.

4. Содержание отчета

Наименование и цель работы. Краткая основная теоретическая часть, которая применялась вами при выполнении работы. Комментарии, а также снимки экрана для каждого пункта лабораторной работы, которые отражали бы весь ход выполнения. Вывод о полученных навыках и знаниях после выполнения работы.

Контрольные вопросы для самопроверки

1. Что такое исключение?
2. Как производится обработка исключения?
3. Для чего предназначена инструкция try?
4. Для чего предназначена инструкция catch?
5. Для чего предназначена инструкция throw?
6. Может ли обработчик исключительной ситуации быть объявлен за пределами функции?
7. Может ли обработчик исключительной ситуации быть объявлен вне класса?

Библиотека ГГТУ им. П.О. Сухого

Лабораторная работа № 9

Основы построения графического интерфейса, базовый класс QWidget. Система сигналов и слотов

1. Цель работы

Изучить принципы построения графического интерфейса.

2. Основные теоретические сведения

2.1. Класс QWidget

Класс QWidget унаследован от класса QObject, а значит, может использовать механизм сигналов/слотов и механизм объектной иерархии. Благодаря этому виджеты могут иметь потомков, которые отображаются внутри предка. Это очень важно, так как каждый виджет может служить контейнером для других виджетов, – т. е. в Qt нет разделения между элементами управления и контейнерами. Виджеты в контейнерах могут выступать в роли контейнеров для других виджетов, и так до бесконечности. Например, диалоговое окно содержит кнопки Ok и Cancel (Отмена) – следовательно, оно является контейнером. Это удобно еще и потому, что если виджет-предок станет недоступным или невидимым, то виджеты-потомки автоматически примут его состояние.

Класс Qwidget и большинство унаследованных от него классов имеют конструктор с двумя параметрами:

QWidget(QWidget* pwgt = 0, Qt::WindowFlags f = 0)

Например:

wgt.setWindowFlags(Qt::Window | Qt::WindowTitleHint |
Qt::WindowStaysOnTopHint);

При помощи слот-метода setwindowTitle () устанавливается надпись заголовка окна. Но это имеет смысл только для виджетов верхнего уровня.

Например:

wgt.setwindowTitle("My Window");

Слот setEnabled () устанавливает виджет в доступное (enabled) или недоступное (disabled) состояние. Параметр true соответствует доступному, а false — недоступному состоянию. Чтобы узнать, в каком состоянии находится виджет, вызовите метод isEnabled ().

Библиотека ГГТУ им.П.О.Сухого

На рис. 9.1 изображен внешний вид различных типов окон с различными модификаторами.

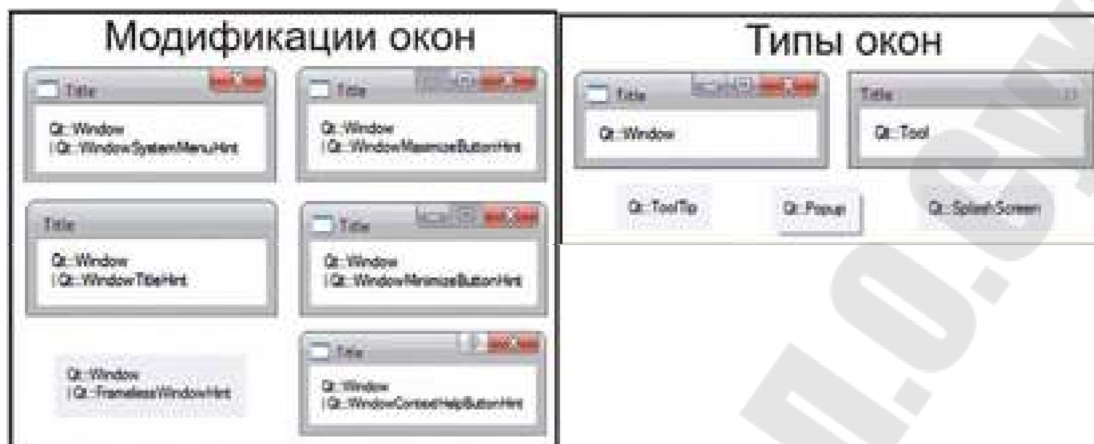


Рис. 9.1. Вид окон виджетов верхнего уровня

2.2. Механизм сигналов и слотов

Проблема расширения языка С++ решена в Qt с помощью специального препроцессора МОС (Meta Object Compiler, метаобъектный компилятор). Он анализирует классы на наличие в их определении специального макроса Q OBJECT и внедряет в отдельный файл всю необходимую дополнительную информацию. Это происходит автоматически, без непосредственного участия разработчика.

В программировании с использованием Qt под понятием сигналы подразумеваются методы, которые в состоянии осуществлять пересылку сообщений.

Сигналы определяются в классе, как и обычные методы, только без реализации. С точки зрения программиста они являются лишь прототипами методов, содержащихся в заголовочном файле определения класса. Всю дальнейшую заботу о реализации кода для этих методов берет на себя МОС. Методы сигналов не должны возвращать каких-либо значений, и поэтому перед именем метода всегда должен стоять возвращаемый параметр void.

Библиотека предоставляет большое количество уже готовых сигналов для существующих элементов управления. В основном, для решения поставленных задач хватает этих сигналов, но иногда возникает необходимость реализации новых сигналов в своих классах.

Из сказанного становится ясно, что не имеет смысла определять сигналы как private, protected или public, поскольку они играют роль вызывающих методов.

Библиотека ГГТУ им.П.О.Сухого

Слоты (slots) – это методы, которые присоединяются к сигналам. По сути, они являются обычными методами. Самое большое их отличие состоит в возможности принимать сигналы. Как и обычные методы, они определяются в классе как public, private или protected. Если необходимо сделать так, чтобы слот мог соединяться только с сигналами сторонних объектов, но не вызываться как обычный метод со стороны, то тогда слот нужно объявить как protected или private. Во всех других случаях объявляйте их как public. В объявлениях перед каждой группой слотов должно стоять соответственно: private slots:, protected slots: или public slots:. Слоты могут быть и виртуальными.

Соединение объектов осуществляется при помощи статического метода connect (), который определен в классе QObject. В общем виде вызов метода connect () выглядит следующим образом:

```
QObject::connect(const QObject* sender,
const char* signal,
const QObject* receiver,
const char* slot,
Qt::ConnectionType type = Qt:Autoconnection);
```

Ему передаются пять следующих параметров:

- sender – указатель на объект, отправляющий сигнал;
- signal – это сигнал, с которым осуществляется соединение. Прототип (имя и аргументы) метода сигнала должен быть заключен в специальный макрос SIGNAL (method ());
- receiver – указатель на объект, который имеет слот для обработки сигнала;
- slot – слот, который вызывается при получении сигнала. Прототип слота должен быть заключен в специальный макрос slot (method ());
- type – управляет режимом обработки. Имеется три возможных значения: Qt::DirectConnection – сигнал обрабатывается сразу вызовом соответствующего метода слота, Qt::QueuedConnection – сигнал преобразуется в событие и ставится в общую очередь для обработки, Qt:Autoconnection – это автоматический режим, который действует следующим образом: если отсылающий сигнал объект находится в одном потоке с принимающим его объектом, то устанавливается режим Qt::DirectConnection. В противном случае – режим Qt::QueuedConnection. Этот режим (Qt:Autoconnection) определен в методе connection () по умолчанию. Вам вряд ли придется изменять режимы «вручную», но полезно знать, что такая возможность есть.

Библиотека ГГТУ им. П.О.Сухого

Если есть возможность соединения объектов, то должна существовать и возможность их разъединения. В Qt при уничтожении объекта все связанные с ним соединения уничтожаются автоматически, но в редких случаях может возникнуть необходимость в уничтожении этих соединений «вручную». Для этого существует статический метод disconnect (), параметры которого аналогичны параметрам статического метода connect (). В общем виде этот метод выглядит таким образом:

```
QObject::disconnect(sender, signal, receiver, slot);
```

3. Порядок выполнения работы

3.1. Создание графического интерфейса на основе QWidget и его потомков

Рассмотрим пример создания окна приложения на основе класса текстовой метки (QLabel). Проект состоит из двух файлов Programm9_1.pro и main.cpp.

Листинг программы:

Файл Programm9_1.pro

```
QT += core gui
greaterThan(QT_MAJOR_VERSION, 4): QT += widgets
TARGET = Programm9_1
TEMPLATE = app
SOURCES += main.cpp
```

Файл main.cpp

```
#include <QLabel>
#include <QApplication>
int main(int argc, char *argv[]) {
  QApplication a(argc, argv); //Передаем классу QApplication параметры запуска программы
  QLabel w; //Создам виджет верхнего уровня
  w.setText("Hello world"); //Устанавливаем текст
  w.show(); //Делаем виджет видимым
  return a.exec();
}
```

Рассмотрим более подробно файл Programm9_1.pro. Опция QT += указывает, какие модули должны быть подключены к проекту. При этом += означает то, что к уже подключенным модулям добавляются указанные в этой строке. В данном примере подключаются модули core, gui и widgets. Опция TARGET задает имя исполняемого

Библиотека ГГТУ им.П.О.Сухого

файла, который получится в результате сборки проекта. В данном примере (TARGET = Programm9_1) мы получим исполняемый файл Programm9_1.exe. Параметр TEMPLATE указывает на то, что данный проект является приложением (в результате сборки должен получиться исполняемый файл), а не библиотека или модуль. Опция SOURCES задает список файлов с исходным кодом включенных в проект. В данном случае к проекту относится только один файл main.cpp, который располагается в той же папке, что и файл проекта.

При выполнении программы будет создано графическое окно, содержащее текст Hello world.

3.2. Создание окна на основе собственного потомка класса QWidget

В данном примере будет объявлен класс-потомок от QWidget, и на основе его будет создано основное окно программы (виджет верхнего уровня). Рассмотрим листинг программы.

Файл Programm9_2.pro

```
QT       += core gui
greaterThan(QT_MAJOR_VERSION, 4): QT += widgets
TARGET = Programm9_2
TEMPLATE = app
SOURCES += main.cpp\
        widget.cpp
HEADERS  += widget.h
```

Файл widget.h

```
#ifndef WIDGET_H
#define WIDGET_H
#include <QWidget>
#include <QDebug>
#include <QLabel>
class Widget : public QWidget {
   Q_OBJECT
public:
   Widget(QWidget *parent = 0);
   ~Widget();
private:
   QLabel *label1=NULL;
};
#endif // WIDGET_H
```

Файл widget.cpp

```
#include "widget.h"
Widget::Widget(QWidget *parent) : QWidget(parent){
   qDebug()<<"Create window";
   this->setWindowTitle("Main Window");
```

Файл main.cpp

```
#include "widget.h"
#include <QApplication>
int main(int argc, char *argv[]) {
   QApplication a(argc, argv);
   Widget w;
   w.show();
```


Библиотека ГГТУ им. П.О.Сухого

```
  label1 = new QLabel("Hello world",this);
}
Widget::~Widget(){
  qDebug()<<"Destroy window";
}
```

```
  return a.exec();
}
```

В файле проекта присутствуют аналогичные параметры, как и в предыдущем примере, за исключением опции HEADERS. Она задает список заголовочных файлов, относящихся к проекту.

Заголовочный файл **widget.h** содержит описание структуры класса, наследника от QWidget, на основе которого создается основное окно программы. Также данный класс является контейнером, так как содержит в себе объект класса QLabel.

В конструкторе класса основного окна производим следующие действия:

– выводим текстовое сообщение в отладочную консоль программы (qDebug()<<"Create window";);

– устанавливаем текст заголовка окна (this->setWindowTitle-("Main Window"););

– создаем виджет текстовой метки (label1 = new QLabel("Hello world",this);).

В деструкторе класса содержится только вывод информации в отладочную консоль приложения (qDebug()<<"Destroy window";).

В результате запуска приложения будет создано окно, с заголовком Main Window и содержащее текстовую метку. При выполнении конструктора класса (при создании окна) в отладочную консоль будет выдано сообщение Create window, а при закрытии окна будет вызван деструктор и в отладочной консоли появится сообщение Destroy window.

3.3. Система сигналов и слотов

Файл Programm9_3.pro

```
QT       += core gui
TARGET = Programm9_3
TEMPLATE = app
SOURCES += main.cpp\
      widget.cpp
HEADERS  += widget.h
```

Файл widget.h

```
#ifndef WIDGET_H
#define WIDGET_H
#include <QWidget>
#include <QPushButton>
#include <QDebug>
class Widget : public QWidget {
   Q_OBJECT
public:
```


Библиотека ГГТУ им. П.О.Сухого

```
    Widget(QWidget *parent = 0);
    ~Widget();
private:
    QPushButton *button=NULL;
    quint16 pressCount=0;
private slots:
    void slotPressButton();
};
#endif // WIDGET_H
```

Файл widget.cpp

```
#include "widget.h"
Widget::Widget(QWidget *parent) : QWidget(parent) {
    button = new QPushButton(tr("Press me"),this);
    con-
nect(button,SIGNAL(pressed()),this,SLOT(slotPressButton()));
}
Widget::~Widget() {
}
void Widget::slotPressButton() {
    qDebug()<<"Button pressed "<<QString::number(pressCount);
    pressCount++;
}
```

Файл main.cpp

```
#include "widget.h"
#include <QApplication>
int main(int argc, char *argv[]) {
    QApplication a(argc, argv);
    Widget w;
    w.show();
    return a.exec();
}
```

В данном примере практически все аналогично предыдущему, кроме того, что в качестве дочернего виджета используется кнопка. Сигнал кнопки подключается к слоту, расположенному в классе основного окна. При нажатии происходит инкремент переменной счетчика и в отладочную консоль приложения выводится надпись.

Библиотека ГГТУ им.П.О.Сухого

3.4. Задание для самостоятельной работы

Файл содержит несколько строк, в каждой из которых записано единственное выражение вида a#b (без ошибок), где a, b – целочисленные величины, # – операция +, –, /, *. Написать программу, которая при нажатии на кнопку считывает эти строки, выводит выражения в отладочную консоль приложения, а также производит их вычисления. Работу с файлами реализовать с использованием классов Qt.

4. Содержание отчета

Наименование и цель работы. Краткая основная теоретическая часть, которая применялась вами при выполнении работы. Комментарии, а также снимки экрана для каждого пункта лабораторной работы, которые отражали бы весь ход выполнения. Вывод о полученных навыках и знаниях после выполнения работы.

Контрольные вопросы для самопроверки

1. Что такое QWidget?
2. В чем особенность виджета верхнего уровня?
3. Каким образом происходит создание графического интерфейса?
4. Что такое сигнал?
5. Что такое слот?
6. Соединение сигналов и слотов.
7. Какие параметры может иметь функция connect?

Библиотека ГГТУ им. П.О. Сухого

Лабораторная работа № 10

Управление компоновкой элементов на форме

1. Цель работы

Изучить принципы автоматической компоновки элементов на форме.

2. Основные теоретические сведения

От класса QLayout унаследованы классы QGridLayout и QBoxLayout (рис. 10.1). Класс QGridLayout управляет табличным размещением, а от **QBoxLayout** унаследованы два класса QHBoxLayout и QVBoxLayout для горизонтального и вертикального размещения.

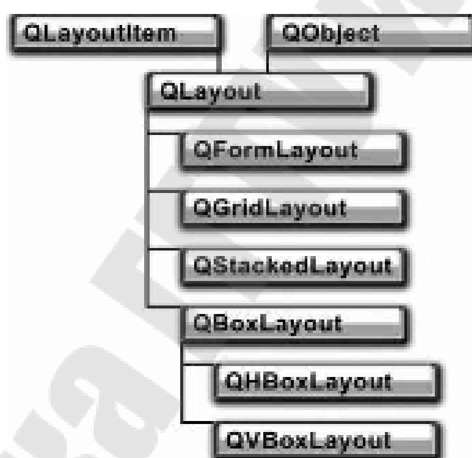


Рис. 10.1. Иерархия классов менеджеров компоновки

2.1. Класс QBoxLayout

Объект класса QBoxLayout может управлять как горизонтальным, так и вертикальным размещением. Для того чтобы задать способ размещения, первым параметром конструктора должно быть одно из следующих значений:

- LeftToRight – горизонтальное размещение, заполнение осуществляется слева направо;
- RightToLeft – горизонтальное размещение, заполнение выполняется справа налево;

Библиотека ГГТУ им. П.О.Сухого

• TopToBottom – вертикальное размещение, заполнение осуществляется сверху вниз;

• BottomToTop – вертикальное размещение, заполнение выполняется снизу вверх.

К компоновке при помощи метода **addSpacing**() можно добавить заданное расстояние между двумя виджетами.

Класс QBoxLayout определяет свой собственный метод addwidget() для добавления виджетов в компоновку с возможностью указания в дополнительном параметре фактора растяжения (по умолчанию этот параметр равен нулю).

2.2. Горизонтальное размещение QHBoxLayout

Объекты класса QHBoxLayout упорядочивают все виджеты только в горизонтальном порядке – слева направо (рис. 10.2). Его применение аналогично использованию класса QBoxLayout, но передавать в конструктор дополнительный параметр, задающий горизонтальный порядок размещения, не нужно.

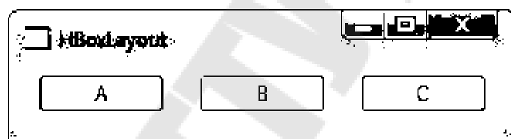


Рис. 10.2. Размещение кнопок по горизонтали

2.3. Вертикальное размещение QVBoxLayout

Компоновка QVBoxLayout унаследована от QBoxLayout и упорядочивает все виджеты только по вертикали – сверху вниз (рис. 10.3). В остальном она ничем не отличается от классов QBoxLayout и QHBoxLayout.

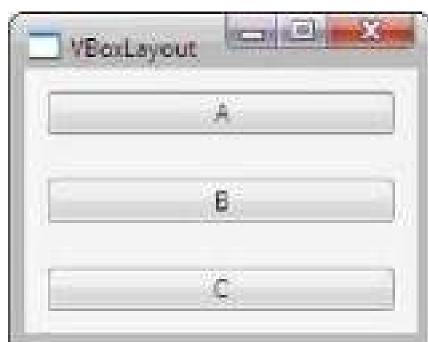


Рис. 10.3. Размещение кнопок по вертикали

Библиотека ГГТУ им. П.О.Сухого

Факторы растяжения можно самостоятельно добавлять в компоновки, для чего существует метод addstretch(). В этом случае фактор растяжения образно можно сравнить с пружиной, которая находится между виджетами и может иметь различную упругость в соответствии с задаваемым параметром. На рис. 10.4 показан пример добавления фактора растяжения между двумя виджетами для расположения виджетов по краям окна.

Рис. 10.4. Фактор растяжения

2.4. Вложенные размещения

Размещая одну компоновку внутри другой, можно создавать размещения практически любой сложности. Для организации вложенных размещений существует метод addLayout, в котором вторым параметром передается фактор растяжения для добавляемой компоновки.

На рис. 10.5 показан пример вложенного размещения двух менеджеров компоновки. В компоновку QVBoxLayout помещается компоновка QHBoxLayout.

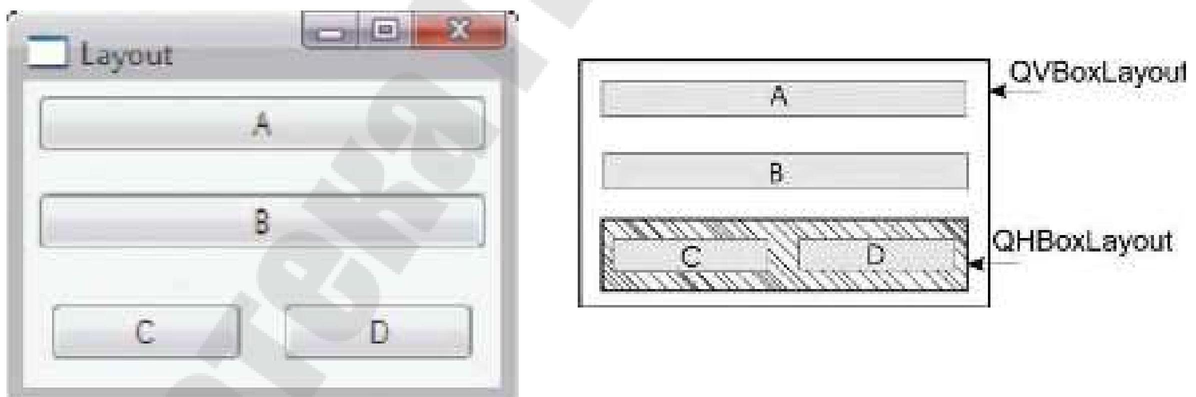


Рис. 10.5. Вложенное размещение

2.5. Табличное размещение QGridLayout

Для табличного размещения используется класс QGridLayout, с помощью которого можно быстро создавать сложные по структуре размещения. Таблица состоит из ячеек, позиции которых задаются строками и столбцами.

Библиотека ГГТУ им. П.О.Сухого

Добавить виджет в таблицу можно с помощью метода **addwidget** (), передав ему позицию ячейки, в которую помещается виджет. Иногда необходимо, чтобы виджет занимал сразу несколько позиций, чего можно достичь тем же методом **addwidget()**, указав в дополнительных параметрах количество строк и столбцов, которые будет занимать виджет. В последнем параметре можно задать выравнивание (табл. 10.1), например, по центру:
playout->addWidget(widget, 17, 1, Qt::AlignCenter);

Таблица 10.1

Значения флагов Alignment Flag пространства имен Qt

Константа	**Значение**	**Описание**
AlignLeft	0x0001	Расположение текста слева
AlignRight	0x0002	Расположение текста справа
AlignHCenter	0x0004	Центровка текста по горизонтали
AlignJustify	0x0008	Растягивание текста по всей ширине
AlignTop	0x0010	Расположение текста вверху
AlignBottom	0x0020	Расположение текста внизу
AlignVCenter	0x0040	Центровка текста по вертикали
AlignCenter	AlignVCenter | AlignHCenter	Центровка текста по вертикали и горизонтали

Фактор растяжения устанавливается методами setRowStretch () и setColumnStretch (), но не для каждого виджета в отдельности, а для строки или столбца. Расстояние между виджетами также устанавливается для столбцов или строк методом setspacing ().

Пример, показанный на рис. 10.6, размещает четыре кнопки: А, В, С и D в таблице размером 2 на 2 ячейки.

Рис. 10.6. Размещение кнопок в табличном порядке

2.6. Порядок следования табулятора

Пользователь может взаимодействовать с виджетами при помощи мыши и клавиатуры. В последнем случае для выбора нужного

Библиотека ГГТУ им. П.О. Сухого

виджета используется клавиша табуляции – <TaЬ>, при нажатии которой происходит переход фокуса согласно установленному порядку от одного виджета к другому. Иногда возникает необходимость в изменении этого порядка, который по умолчанию соответствует очередности установки дочерних виджетов в виджете предка. На рис. 10.4 цифрами изображен порядок смены фокуса с помощью табулятора. При появлении диалогового окна с тремя кнопками фокус будет установлен на кнопке C и после нажатия на клавишу табуляции он перейдет на кнопку B, а затем — на кнопку A (рис. 10.7).

а) *б*)

Рис. 10.7. Порядок смены фокуса: *а* – прямой; *б* – измененный

Изменить порядок смены фокуса можно с помощью статического метода **QWidget:: setTabOrder** (), получающего в качестве параметров два указателя на виджеты. Следующие вызовы изменят порядок следования табулятора на более логичный порядок:

```
QWidget::setTabOrder(A, B);
QWidget::setTabOrder(B, C);
```

3. Порядок выполнения работы

Рассмотрим несколько примеров программ, которые демонстрируют основные возможности автоматической компоновки элементов на форме.

3.1. Компоновка с помощью QBoxLayout

Данный пример реализует компоновку шести кнопок на основной форме в два ряда. Первый ряд располагает элементы слева направо, а второй справа налево.

Файл Programm10_1.pro

```
QT       += core gui
TARGET = Programm10_1
```

Файл main.cpp

```
#include "widget.h"
#include <QApplication>
```


Библиотека БГУИР

```
TEMPLATE = app
SOURCES += main.cpp\
                 widget.cpp
HEADERS  += widget.h
```

```
int main(int argc, char *argv[]) {
    QApplication a(argc, argv);
    Widget w;
    w.show();
    return a.exec();
}
```

Файл widget.h

```
#ifndef WIDGET_H
#define WIDGET_H
#include <QWidget>
#include <QBoxLayout>
#include <QPushButton>
class Widget : public QWidget {
    Q_OBJECT
public:
    Widget(QWidget *parent = 0);
    ~Widget();
private:
    QBoxLayout *mainLayout=NULL;
    QBoxLayout *layout1=NULL;
    QPushButton *buttonA=NULL;
    QPushButton *buttonB=NULL;
    QPushButton *buttonC=NULL;
    QBoxLayout *layout2=NULL;
    QPushButton *buttonD=NULL;
    QPushButton *buttonE=NULL;
```

Файл widget.cpp

```
#include "widget.h"
Widget::Widget(QWidget *parent) : QWidget(parent) {
    mainLayout = new QBoxLayout(QBoxLayout::TopToBottom,this);
    layout1 = new QBoxLayout(QBoxLayout::LeftToRight);
    mainLayout->addLayout(layout1);
    buttonA = new QPushButton(tr("Button A"),this);
    layout1->addWidget(buttonA);
    buttonB = new QPushButton(tr("Button B"),this);
    layout1->addWidget(buttonB);
    buttonC = new QPushButton(tr("Button C"),this);
    layout1->addWidget(buttonC);
    layout2 = new QBoxLayout(QBoxLayout::RightToLeft);
    mainLayout->addLayout(layout2);
    buttonD = new QPushButton(tr("Button D"),this);
    layout2->addWidget(buttonD);
    buttonE = new QPushButton(tr("Button E"),this);
    layout2->addWidget(buttonE);
    buttonF = new QPushButton(tr("Button F"),this);
    layout2->addWidget(buttonF);
}
```

```
  QPushButton
*buttonF=NULL;
};
#endif // WIDGET_H
```

```
Widget::~Widget() {
   delete buttonA;
   delete buttonB;
   delete buttonC;
   delete buttonD;
   delete buttonE;
   delete buttonF;
   delete layout1;
   delete layout2;
}
```

Файлы Programm10_1.pro и main.cpp отдельно рассматривать не стоит, они типичны и хорошо описаны в предыдущих лабораторных.

Рассмотрим класс основного окна программы. В начале файла widget.h производим подключение необходимых библиотек для использования следующих классов:

– QWidget – класс-родитель для класса основного окна;

– QBoxLayout – класс автоматической компановки элементов на форме;

– QPushButton – класс кнопки.

Теперь рассмотрим конструктор класса основного окна программы. Сначала создается основной объект копоновки:

mainLayout = new QBoxLayout(QBoxLayout::TopToBottom,this);

Элементы будут на нем располагаться вертикально и порядок заполнения будет сверху вниз (за это отвечает опция QBoxLayout::TopToBottom, переданная в конструктор). Нам не следует явно задавать, что данный объект будет основным для виджета окна, так как в параметрах указателя основное окно программы передано в качестве родителя.

Затем создается объект компоновки первой строки виджетов и укладывается на основной менеджер компоновки:

layout1 = new QBoxLayout(QBoxLayout::LeftToRight);
mainLayout->addLayout(layout1);

Виджеты, которые будут уложены на него, будут располагаться горизонтально, с порядком заполнения слева направо.

Аналогичным образом поступаем и со вторым объектом компоновки:

layout2 = new QBoxLayout(QBoxLayout::RightToLeft);
mainLayout->addLayout(layout2);

Только в отличии от первого порядок заполнения элементами у

Библиотека ГГТУ им.П.О.Сухого

него установлен справо налево.

Затем создаем кнопки и укладываем их на соответствующие объекты компоновки:

```
buttonA = new QPushButton(tr("Button A"),this);
layout1->addWidget(buttonA);
.
.
.
buttonF = new QPushButton(tr("Button F"),this);
layout2->addWidget(buttonF).
```

В результате выполнения данной программы мы получим форму, на которой виджеты (кнопки) расположены в два ряда. В первом ряду порядок кнопок будет слева направо, а во втором наоборот.

3.2. Компоновка с помощью QVBoxLayout и QHBoxLayout

Рассмотрев более общий класс компоновки виджетов, рассмотрим специализированные **QVBoxLayout** и **QHBoxLayout**. Как уже было сказано, в теоретической части первый класс компоновки осуществляет вертикальное расположение виджетов, а второй – горизонтальное. Пример применения данных классов для размещения виджетов на форме:

Файл Programm10_2.pro

```
QT       += core gui
TARGET = Programm10_2
TEMPLATE = app
SOURCES += main.cpp\
       widget.cpp
HEADERS  += widget.h
```

Файл main.cpp

```
#include "widget.h"
#include <QApplication>
int main(int argc, char *argv[]) {
   QApplication a(argc, argv);
   Widget w;
   w.show();
   return a.exec();
}
```

Файл widget.h

```
#ifndef WIDGET_H
#define WIDGET_H
#include <QWidget>
#include <QHBoxLayout>
#include <QVBoxLayout>
```

Файл widget.cpp

```
#include "widget.h"
Widget::Widget(QWidget *parent) :
QWidget(parent) {
   mainLayout = new QVBoxLay-
out(this);
```


Библиотека ГГТУ им. П.О. Сухого

```
#include <QLabel>
#include <QLineEdit>
#include <QPushButton>
class Widget : public Qwidget {
   Q_OBJECT
public:
   Widget(QWidget *parent = 0);
   ~Widget();
private:
   QVBoxLayout
*mainLayout=NULL;
   QHBoxLayout *layout1=NULL;
   QLabel *labelA=NULL;
   QLineEdit *editA=NULL;
   QHBoxLayout *layout2=NULL;
   QPushButton
*buttonSetA=NULL;
};
#endif // WIDGET_H
```

```
   layout1 = new QHBoxLayout;
   mainLayout-
>addLayout(layout1);
   labelA = new QLabel(tr("Enter
A"),this);
   layout1->addWidget(labelA);
   editA = new QLineEdit(this);
   layout1->addWidget(editA);
   layout2 = new QHBoxLayout;
   mainLayout-
>addLayout(layout2);
   layout2->addStretch();
   buttonSetA = new QPushBut-
ton(tr("Set A"),this);
   layout2-
>addWidget(buttonSetA);
}
Widget::~Widget() {
}
```

Как обычно, в заголовочном файле «класс основного окна» произведем подключение необходимых библиотек для данного проекта:

– QWidget – класс-родитель класса основного окна;
– QHBoxLayout – горизонтальный компоновщик виджетов;
– QVBoxLayout – вертикальный компоновщик виджетов;
– QLabel – класс текстовой метки;
– QLineEdit – класс текстового поля;
– QPushButton – класс кнопки.

В структуре класса основного окна произведем объявление указателей на необходимые объекты.

В конструкторе класса основного окна произведем создание объектов и расположение их на форме.

Сначала создаем основной компоновщик виджетов:

```
mainLayout = new QVBoxLayout(this);
```

Затем создаем компоновщики первой и второй строки и располагаем их на основном компоновщике:

```
layout1 = new QHBoxLayout;
```


Библиотека ГГТУ им. П.О.Сухого

```
mainLayout->addLayout(layout1);
layout2 = new QHBoxLayout;
mainLayout->addLayout(layout2);
```

Создаем элементы (виджеты) и располагаем их на нужных компоновщиках. В первой строке расположим текстовую метку и текстовое поле. Их создание типичное, и не требует дополнительных пояснений. Во второй строке расположим кнопку. Однако необходимо сделать так, чтобы при изменении ширины формы кнопка не меняла своего размера.

```
layout2->addStretch();
buttonSetA = new QPushButton(tr("Set A"),this);
layout2->addWidget(buttonSetA);
```

Для этого перед расположение формы создадим фактор растяжения (layout2->addStretch()), а только после этого создадим объект кнопки и расположим его на компоновщике.

3.3. Компоновка с помощью QGridLayout

Данный метод компоновки элементов на форме позволяет выравнивать положение виджетов по сетке. Рассмотрим пример:

Файл Programm10_3.pro

```
QT       += core gui
TARGET = Programm10_3
TEMPLATE = app
SOURCES += main.cpp\
      widget.cpp
HEADERS  += widget.h
```

Файл main.cpp

```
#include "widget.h"
#include <QApplication>
int main(int argc, char *argv[]) {
   QApplication a(argc, argv);
   Widget w;
   w.show();
   return a.exec();
}
```

Файл widget.h

```
#ifndef WIDGET_H
#define WIDGET_H
#include <QWidget>
#include <QGridLayout>
#include <QLabel>
#include <QLineEdit>
#include <QPushButton>
class Widget : public QWidget {
```

Файл widget.cpp

```
#include "widget.h"
Widget::Widget(QWidget *parent) :
QWidget(parent) {
   mainLayout = new QGridLay-
out(this);
   labelA = new QLabel(tr("Enter
A"),this);
   mainLayout-
```


Библиотека ГГТУ им. П.О. Сухого

```
  Q_OBJECT
public:
  Widget(QWidget *parent = 0);
  ~Widget();
private:
  QGridLayout
*mainLayout=NULL;
  QLabel *labelA=NULL;
  QLineEdit *editA=NULL;
  QLabel *labelB=NULL;
  QLineEdit *editB=NULL;
  QPushButton *buttonSet=NULL;
};
#endif // WIDGET_H
```

```
>addWidget(labelA,0,0,1,1);
  editA = new QLineEdit(this);
  mainLayout-
>addWidget(editA,0,1,1,2);
  labelB = new QLabel(tr("Enter
B"),this);
  mainLayout-
>addWidget(labelB,1,0,1,1);
  editB = new QLineEdit(this);
  mainLayout-
>addWidget(editB,1,1,1,2);
  buttonSet = new QPushBut-
ton(tr("Set"),this);
  mainLayout-
>addWidget(buttonSet,2,2,1,1);
}
Widget::~Widget(){
}
```

Принцип использования QGridLayout аналогичен предыдущим, однако при расположении виджета необходимо указывать дополнительные параметры.

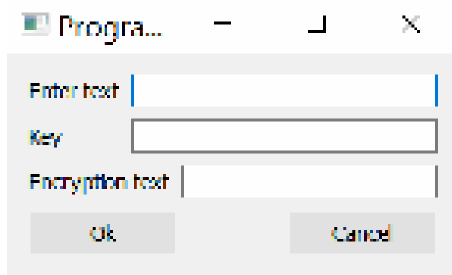
mainLayout->addWidget(labelA,0,0,1,1);

Первый параметр, как и у всех компоновщиков, является указателем на виджет. Затем обязательно указывается строка и столбец, в котором необходимо расположить элемент, а также количество ячеек, которые должен занимать виджет по горизонтали и вертикали. В данном случае виджет labelA располагается в 0 строке, в 0 столбце и занимает по одной ячейке в каждом направлении.

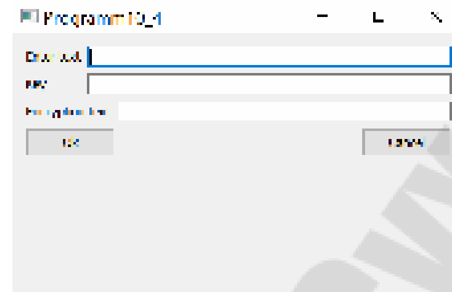
3.4. Задание для самостоятельной работы

Реализовать программу, которая будет создавать форму, аналогичную той, что изображена на рис. 10.8.

Библиотека ГГТУ им. П.О.Сухого



а)



б)

Рис. 10.8. Вид формы окна для задания на самостоятельную работу:
а – исходное состояние формы; *б* – вид формы при изменении размера

4. Содержание отчета

Наименование и цель работы. Краткая основная теоретическая часть, которая применялась вами при выполнении работы. Комментарии, а также снимки экрана для каждого пункта лабораторной работы, которые отражали бы весь ход выполнения. Вывод о полученных навыках и знаниях после выполнения работы.

Контрольные вопросы для самопроверки

1. Что такое классы управления автоматического размещения элементов?
2. В чем заключаются особенности применения QBoxLayout?
3. В чем заключаются особенности применения QHBoxLayout?
4. В чем заключаются особенности применения QVBoxLayout?
5. В чем заключаются особенности применения QGridLayout?
6. Что такое фактор растяжения?
7. Как задается порядок следования табулятора?

Библиотека ГГТУ им. П.О. Сухого

Лабораторная работа № 11

Работа с элементами отображения

1. Цель работы

Изучить основные элементы отображения и методы их использования при построении формы.

2. Основные теоретические сведения

Элементы отображения не принимают активного участия в действиях пользователя и используются для информирования его о происходящем. Эта информация может носить как текстовый, так и графический характер (картинки, графика).

2.1. Надписи (Qlabel)

Виджет надписи служит для показа состояния приложения или поясняющего текста и представляет собой текстовое поле, текст которого не подлежит изменению со стороны пользователя. Информация, отображаемая этим виджетом, может изменяться только самим приложением. Таким образом, приложение может сообщить пользователю о своем изменившемся состоянии, но пользователь не может изменить эту информацию в самом виджете. Класс виджета надписи QLabel определен в заголовочном файле QLabel:

```
#include <QLabel>
```

Например:

```
QLabel *labelText = new QLabel(tr("Count"),this);
labelText->setAlignment(Qt::AlignVCenter | Qt::AlignHCenter);
```

2.2. Индикатор процесса

Индикатор процесса (progress bar) – это виджет, демонстрирующий процесс выполнения операции и заполняющийся слева направо. Полное заполнение индикатора информирует о завершении операции. Этот виджет необходим в том случае, когда программа выполняет продолжительные действия, – виджет дает пользователю понять, что программа не зависла, а находится в работе. Он также показывает, сколько уже проделано и сколько еще предстоит сделать. Класс QProgressBar виджета индикатора процесса определен в заголовочном

Библиотека ГГТУ им. П.О. Сухого

файле QProgressBar. Обычно индикаторы процесса располагаются в горизонтальном положении, но это можно изменить, передав в слот setOrientation () значение Qt::vertical, – после этого он будет расположен вертикально.

```
QProgressBar  *progres = new QProgressBar();
progres->setMinimum(0); //Задаем минимальное значение
progres->setMaximum(100); //Задаем максимальное значение
progres->setOrientation(Qt::Vertical); //Устанавливаем вертикальную ориентацию.
```

2.3. Электронный индикатор

Класс QLCDNumber виджета электронного индикатора определен в заголовочном файле QLCDNumber. По внешнему виду этот виджет представляет собой набор сегментных указателей — как, например, на электронных часах. С помощью виджета электронного индикатора отображаются целые числа. Допускается использование точки, которую можно отображать между позициями сегментов или как отдельный символ, вызывая метод setSmallDecimalPoint () и передавая в него true или false соответственно. Количество отображаемых сегментов можно задать в конструкторе или с помощью метода setNumDigits (). В том случае, когда для отображения числа не хватает сегментов индикатора, отсылается сигнал overflow ().

По умолчанию стиль электронного индикатора соответствует стилю Outline, но его можно изменить, передав методу setSegmentstyle() одно из следующих значений: QLCDNumber:: Outline, QLCDNumber::-Filled или QLCDNumber::Flat.

3. Порядок выполнения работы

3.1. Вывод текстовой информации в Qlabel

Файл widget.h

```
#ifndef WIDGET_H
#define WIDGET_H
#include <QWidget>
#include <QHBoxLayout>
#include <QLabel>
#include <QPushButton>
class Widget : public QWidget {
  Q_OBJECT
```


Библиотека ГГТУ им. П.О.Сухого

```
public:
   Widget(QWidget *parent = 0);
   ~Widget();
private:
   quint16 Count=0;
   QHBoxLayout *mainLayout=NULL;
   QLabel *labelText=NULL;
   QPushButton *buttonIncr=NULL;
private slots:
   void slotPushButton();
};
#endif // WIDGET_H
```

Файл widget.cpp

```
#include "widget.h"
Widget::Widget(QWidget *parent) : QWidget(parent) {
   mainLayout = new QHBoxLayout(this);
   labelText = new QLabel(tr("Count"),this);
   labelText->setAlignment(Qt::AlignVCenter | Qt::AlignHCenter);
   mainLayout->addWidget(labelText);
   buttonIncr = new QPushButton(tr("Increment"),this);
   connect(buttonIncr,SIGNAL(pressed()),this,SLOT(slotPushButton()));
   mainLayout->addWidget(buttonIncr);
}
Widget::~Widget() {
}
void Widget::slotPushButton() {
   Count++;
   labelText->setText(tr("Count")+"="+QString::number(Count));
}
```

3.2. Использование HTML в QLabel

Файл Programm11_2.pro

```
QT       += core gui
TARGET = Programm11_2
TEMPLATE = app
SOURCES += main.cpp\
      widget.cpp
HEADERS  += widget.h
```

Файл main.cpp

```
#include "widget.h"
#include <QApplication>
int main(int argc, char *argv[]) {
   QApplication a(argc, argv);
   Widget w;
   w.show();
```


Библиотека ГГТУ им.П.О.Сухого

```
    return a.exec();
}
```

Файл widget.cpp

```
#include "widget.h"
Widget::Widget(QWidget *parent) : QWidget(parent) {
   mainLayout = new QVBoxLayout(this);
   labelHTML = new QLabel("<strong>Hello</strong> "
                     "<font color=red>World !",this);
   mainLayout->addWidget(labelHTML);
   labelLink = new QLabel("<CENTER><A
HREF=\"http://www.google.ru\">www.google.ru</A></CENTER>",this);
   labelLink->setOpenExternalLinks(true);
   mainLayout->addWidget(labelLink);
}
Widget::~Widget() {
}
```

3.3. Использование QProgressBar

Файл Programm11_3.pro

```
QT       += core gui
TARGET = Programm11_3
TEMPLATE = app
SOURCES += main.cpp\
       widget.cpp
HEADERS  += widget.h
```

Файл widget.h

```
#ifndef WIDGET_H
#define WIDGET_H
#include <QWidget>
#include <QVBoxLayout>
#include <QHBoxLayout>
#include <QPushButton>
#include <QProgressBar>
class Widget : public QWidget {
   Q_OBJECT
public:
   Widget(QWidget *parent = 0);
```

Файл main.cpp

```
#include "widget.h"
#include <QApplication>
int main(int argc, char *argv[]) {
   QApplication a(argc, argv);
   Widget w;
   w.show();
   return a.exec();}
```

Библиотека ГГТУ им. П.О.Сухого

```
    ~Widget();
private:
    quint8 Position=0;
    QHBoxLayout *mainLayout=NULL;
    QVBoxLayout *layout1=NULL;
    QHBoxLayout *layout2=NULL;
    QProgressBar *progres1=NULL;
    QProgressBar *progres2=NULL;
    QPushButton *buttonIncr=NULL;
    QPushButton *buttonReset=NULL;
private slots:
    void slotPushIncr();
    void slotPushReset();
};
#endif // WIDGET_H
```

Файл widget.cpp

```
#include "widget.h"
Widget::Widget(QWidget *parent) : QWidget(parent) {
    mainLayout = new QHBoxLayout(this);
    layout1 = new QVBoxLayout;
    mainLayout->addLayout(layout1);
    progres1 = new QProgressBar(this);
    progres1->setMinimum(0);
    progres1->setMaximum(100);
    layout1->addWidget(progres1);
    layout2 = new QHBoxLayout;
    layout1->addLayout(layout2);
    buttonIncr = new QPushButton(tr("Increment"),this);
    connect(buttonIncr,SIGNAL(pressed()),this,SLOT(slotPushIncr()));
    layout2->addWidget(buttonIncr);
    buttonReset = new QPushButton(tr("Reset"),this);
    connect(buttonReset,SIGNAL(pressed()),this,SLOT(slotPushReset()));
    layout2->addWidget(buttonReset);
    progres2 = new QProgressBar(this);
    progres2->setMinimum(0);
    progres2->setMaximum(100);
    progres2->setOrientation(Qt::Vertical);
    mainLayout->addWidget(progres2);
```


Библиотека ГГТУ им. П.О. Сухого

```
}
Widget::~Widget() {
}
void Widget::slotPushIncr() {
   Position++;
   progres1->setValue(Position);
   progres2->setValue(Position);
}
void Widget::slotPushReset() {
   Position=0;
   progres1->setValue(0);
   progres2->setValue(0);
}
```

3.4. Применение QLCDNumber

Файл Programm11_4.pro

```
QT       += core gui
TARGET = Programm11_4
TEMPLATE = app
SOURCES += main.cpp\
       mainwindow.cpp
HEADERS  += mainwindow.h
```

Файл main.cpp

```
#include "mainwindow.h"
#include <QApplication>
int main(int argc, char *argv[]) {
   QApplication a(argc, argv);
   MainWindow w;
   w.show();
   return a.exec();
}
```

Файл mainwindow.h

```
#ifndef MAINWINDOW_H
#define MAINWINDOW_H
#include <QWidget>
#include <QVBoxLayout>
#include <QLCDNumber>
#include <QPushButton>
class MainWindow : public QWidget {
   Q_OBJECT
public:
   MainWindow(QWidget *parent = 0);
   ~MainWindow();
private:
   quint16 Count=0;
```

Библиотека ГГТУ им.П.О.Сухого

```
   QVBoxLayout *mainLayout=NULL;
   QLCDNumber *lcdNumber=NULL;
   QPushButton *buttonIncr=NULL;
private slots:
   void slotPushIncr();
};
#endif // MAINWINDOW_H
```

Файл mainwindow.cpp

```
#include "mainwindow.h"
MainWindow::MainWindow(QWidget *parent) : QWidget(parent) {
   mainLayout = new QVBoxLayout(this);
   lcdNumber = new QLCDNumber(this);
   lcdNumber->display(0);
   lcdNumber->setFixedHeight(30);
   mainLayout->addWidget(lcdNumber);
   buttonIncr = new QPushButton(tr("Increment"),this);
   connect(buttonIncr,SIGNAL(pressed()),this,SLOT(slotPushIncr()));
   mainLayout->addWidget(buttonIncr);
}
MainWindow::~MainWindow() {
}
void MainWindow::slotPushIncr() {
   Count++;
   lcdNumber->display(Count);
}
```

3.5. Задание для самостоятельного выполнения

3.5.1. Программа «Таблица умножения»

Реализовать программу, которая по нажатию кнопки производит вывод в Qlabel таблицы умножения.

3.5.2. Программа «Визитная карточка»

Необходимо реализовать программу, которая будет выводить о вас информацию. Фотографию, ФИО, группу и дату рождения вывести в QLabel с использованием html. Ваш возраст (количество полных лет) вывести в QLCDNumber. А с помощью QProgressBar отобразить

Библиотека ГГТУ им. П.О. Сухого

в процентном отношении количество дней, оставшихся до вашего следующего дня рождения, от текущей даты.

4. Содержание отчета

Наименование и цель работы. Краткая основная теоретическая часть, которая применялась вами при выполнении работы. Комментарии, а также снимки экрана для каждого пункта лабораторной работы, которые отражали бы весь ход выполнения. Вывод о полученных навыках и знаниях после выполнения работы.

Контрольные вопросы для самопроверки

1. Что такое класс QLabel и какие основные возможности он имеет?
2. Какой метод QLabel позволяет задавать текст?
3. Поддерживает ли QLabel текст в формате HTML?
4. Каковы основные возможности QProgressBar и какова область его применения?
5. Каковы основные возможности QLCDNumber и какова область его применения?
6. Какие способы вывода форматированного текста в QLabel существуют?
7. Какой метод QLCDNumber позволяет задавать значение для индикации?

Библиотека ГГТУ им. П. О. Сухого

Литература

1. Лафоре, Р. Объектно-ориентированное программирование в С++ / Р. Лафоре. – СПб. : Питер, 2018. – 928 с.

2. Саммерфилд, М. Qt профессиональное программирование, разработка кроссплатформенных приложений на С++ / М. Саммерфилд. – М. : Символ, 2011. – 533 с.

3. Боровский, А. Н. Qt4.7 Практическое программирование на С++ / А. Н. Боровский. – СПб. : БВХ-Петербург, 2012. – 492 с.

4. Лаптев, В. В. С++ объектно-ориентированное программирование / В. В. Лаптев. – СПб. : Питер, 2008. – 464 с.

5. Кьоу, Д. Объектно-ориентированное программирование / Д. Кьоу. – СПб. : Питер, 2005. – 240 с.

6. Шлее, М. Qt 5.3. Профессиональное программирование на С++ / М. Шлее. – СПб. : БХВ-Петербург, 2015. – 928 с.

Библиотека ГГТУ им. П.О.Сухого

Содержание

Лабораторная работа № 5. Наследование классов, механизм виртуальных функций, переопределение операций 3
Лабораторная работа № 6. Программирование шаблонов функций и классов 11
Лабораторная работа № 7. Файловый ввод-вывод с применением файловых потоков 17
Лабораторная работа № 8. Функции обработки исключительных ситуаций 27
Лабораторная работа № 9. Основы построения графического интерфейса, базовый класс QWidget. Система сигналов и слотов 33
Лабораторная работа № 10. Управление компоновкой элементов на форме 41
Лабораторная работа № 11. Работа с элементами отображения 53
Литература 61

Библиотека ГГТУ им. П.О.Сухого

Учебное электронное издание комбинированного распространения

Учебное издание

Сахарук Андрей Владимирович

ТЕХНОЛОГИЯ РАЗРАБОТКИ ПРОГРАММНОГО ОБЕСПЕЧЕНИЯ СИСТЕМ УПРАВЛЕНИЯ: ФУНКЦИИ И КЛАССЫ. ЭЛЕМЕНТЫ ОТОБРАЖЕНИЯ

Практикум
по выполнению лабораторных работ
по дисциплине «Технология разработки
программного обеспечения систем управления»
для студентов специальности 1-53 01 07
«Информационные технологии и управление
в технических системах»
дневной формы обучения

Электронный аналог печатного издания

Редактор *Н. В. Гладкова*
Компьютерная верстка *И. П. Минина*

Подписано в печать 19.11.18.
Формат 60х84/$_{16}$. Бумага офсетная. Гарнитура «Таймс».
Ризография. Усл. печ. л. 3,72. Уч.-изд. л. 3,67.
Изд. № 7.
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