

# AS Computing Practical Project

George Wright  
Candidate Number 2415  
Centre Number 12722

12th May 2005

## Contents

<b>1 Design</b>	<b>3</b>
1.0.1 Background . . . . .	3
1.0.2 Build System . . . . .	3
1.1 Definition of Data Requirements . . . . .	3
1.1.1 Data Types . . . . .	3
1.2 User Interface . . . . .	4
1.2.1 Widgets . . . . .	4
1.2.2 The Main Window . . . . .	5
1.2.3 The New Order Window . . . . .	5
1.2.4 The List Orders Dialogue . . . . .	6
1.3 The Assign Turkeys Window . . . . .	7
<b>2 Record Structure, File Organisation and Processing</b>	<b>8</b>
<b>3 Implementation</b>	<b>13</b>
<b>4 Entity Relationship Diagram</b>	<b>22</b>
<b>5 Security and Integrity of Data</b>	<b>22</b>
5.1 Security . . . . .	22
5.2 Data Integrity . . . . .	22
<b>6 System Design</b>	<b>23</b>
<b>7 Testing</b>	<b>23</b>
7.1 Test Plan . . . . .	23
7.2 Test Data . . . . .	24
7.3 Test Results . . . . .	25
7.4 Data Files . . . . .	28
7.5 Assigning Turkeys . . . . .	34

<b>8 Bibliography</b>	<b>37</b>
<b>9 Miscellaneous Notes</b>	<b>38</b>

# 1 Design

## 1.0.1 Background

The entire application will be written in C++ with Trolltech's Qt widget library as the GUI library. It will also be based upon supporting libraries from the KDE project (<http://www.kde.org>) wherever possible. Qt has a lot of backend classes designed to simplify applications a lot which I will make use of. For example, instead of using the **string** data type from C++, I will use the **QString** class from Qt as it is easier to perform data manipulations using Qt's classes.

For the sake of speed and simplicity, the graphical user interface will be designed in *Qt Designer*.

## 1.0.2 Build System

The application will use the GNU (<http://www.gnu.org>) automake/autoconf build system in order to compile the source code into a machine-executable binary format. The build system template used will be the standard system for applications from the KDE project. The compiler used is the C compiler from the GNU compiler collection (*gcc*). In order to build the program, the following three commands need to be executed within the source directory:

```
$ make -f Makefile.cvs
$ ./configure
$ make
```

The application can then be executed by running the **turkey** binary within the **turkey** subdirectory.

## 1.1 Definition of Data Requirements

As per the specification, the following data is recorded when an order is placed:

- Customer Name
- Contact Telephone Number
- Date of Order
- Turkeys Ordered
- Approximate Weight of Turkey

### 1.1.1 Data Types

Data type ranges are architecture dependant. However, the typical data value ranges are listed wherever possible.

- **int** - Integer; whole numbers between -32768 and +32768

- **float** - Floating point numbers
- **double** - Double precision floating point numbers
- **QString** - Character array
- **QDate** - Container class for storing dates
- **QValueVector** - A dynamic array for storing any data types

The customer name will need to be stored in a data type which can store characters, and hence a **QString** data type will be used.

The contact telephone number will also need to be stored in a data format which can store characters. This is because telephone numbers can contain parentheses and '+' symbols as well as digits, and so it will also be stored in a **QString** class.

The date of order will be stored in a **QDate** class, which is specifically designed for storing dates. This will make the date easier to validate as **QDate** has specific validation functions.

The approximate weight of the turkey will be stored as an **int** in a one-dimensional array in grams instead of kilograms. This is because the weight is to the nearest half kilogram and to store it in grams as an integer would be more memory efficient than to store it in kilograms as a **float** or a **double**.

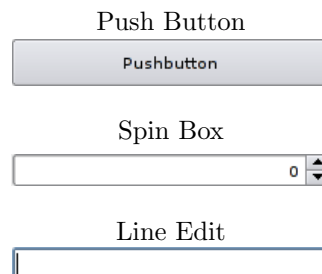
The number of turkeys ordered will be stored as an **int**. This is because only a whole number of turkeys can be ordered.

The order number will be stored as an **int** because it is simply a unique identification number.

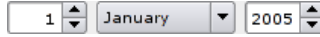
## 1.2 User Interface

### 1.2.1 Widgets

Widgets are the building blocks of any user interface. In this, the following terminology will be used.



### Date Widget



### List View

Col 1	Col 2	
Item	Item	

The user interface will consist of the following:

### 1.2.2 The Main Window

The main window will show the user the current date, the number of turkeys left in the inventory and the total number of turkeys ordered. There will be an option to determine which turkeys are assigned to which order and another option to list all the orders received by the farm thus far. Finally, there will be an option to log a new order into the database.

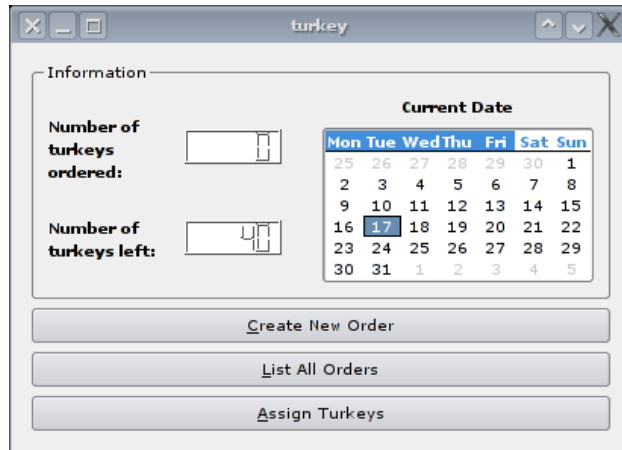


Figure 1: The main window

### 1.2.3 The New Order Window

The new order window will comprise a data entry system which will have a date input box (automatically set to the current date), a name input, a contact telephone number input and a system whereby turkeys can be added to a list with their respective weights.

An estimate of the total cost of the purchase will be shown, as well as broken down costs including the price per turkey, the discount and the preparation surcharge for all the turkeys.

Turkeys can be added to the order by specifying their weight using the spinbox, which has been set up so that it increases in 0.5kg increments, then

clicking on the 'Add Turkey' button which will then add the turkey to the order. Each time a turkey is added to the order, the number of turkeys ordered is incremented by one for the main window.

The confirm push button at the bottom is used to confirm the order.

In the top right is a box showing the order number of the order. This is unique to the order; the first order placed is assigned an order number of '1' and each order thereafter increments the order number by 1. The order can be deleted by pressing the 'Delete Order' button, and can be printed by pressing the 'Print' button.

Weight (kg)	Price (£)	Assigned (grams)
-------------	-----------	------------------

Figure 2: The new order window

#### 1.2.4 The List Orders Dialogue

This dialogue consists solely of a large list view where all the relevant data for each order is shown using different columns. At the bottom of the dialogue is a label which is a total of the profits for all the orders

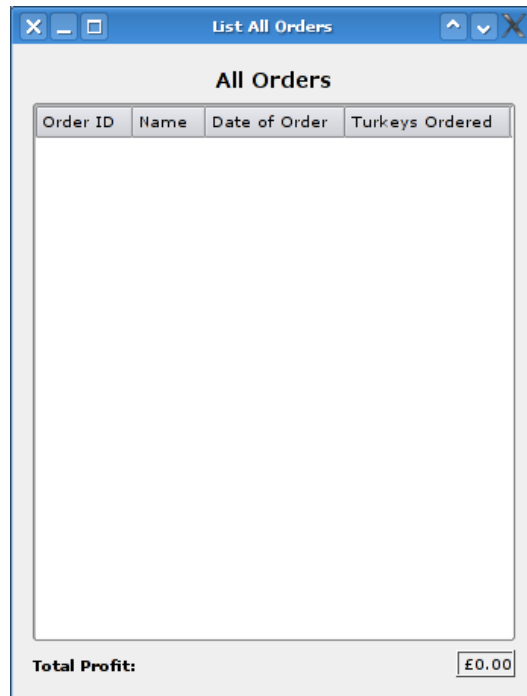


Figure 2: The order listing dialogue

Clicking on one of the orders brings up a dialogue similar to the original ‘New Order’ window, with all the input fields showing the order details of the order. Changes can be made to the order in this dialogue.

The list can be sorted using any of the 5 columns, either in reverse or normal order.

### 1.3 The Assign Turkeys Window

Tools for manipulating the turkeys in the database are in this window. The user can add turkeys to the known inventory using the ‘Add Turkey’ push button. This will take data from the spin box labelled ‘Weight’ and add the turkey to the list, as well as incrementing the number of turkeys left for the main window. It will also require a unique identification for the turkey to be typed in.

In order to delete a turkey from the database because, for example, a turkey has been deemed unfit for consumption, there is a ‘Delete Turkey’ button. A turkey needs to be highlighted in the list before this button is pressed, and that turkey will be deleted from the database.

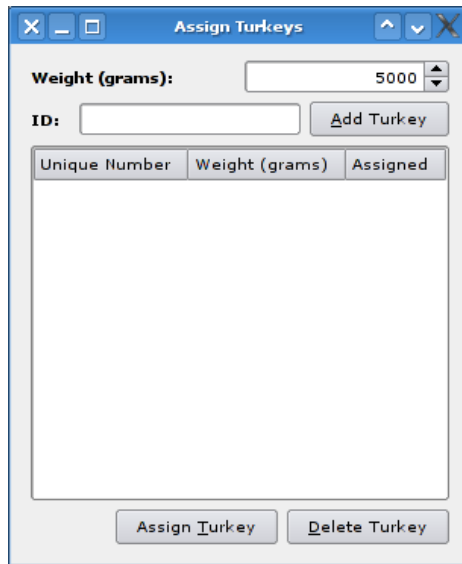


Figure 3: The assign turkeys dialogue

To assign a turkey in the inventory to a customer, an ‘Assign Turkey’ button is provided. The user highlights a turkey in the list, clicks the ‘Assign Turkey’ button, then a dialogue opens which prompts the user to enter the order number of the order the user wishes to add the turkey to. When the turkey is assigned, the ‘No’ in the turkey’s ‘Assigned’ column changes to the number of the order it is assigned to. A turkey can be reassigned using the same method.

## 2 Record Structure, File Organisation and Processing

In memory the data for the program shall be stored in a C-style **struct** defining all the various elements of the order. The **struct** definition is as follows:

```
struct OrderData {
    int orderNumber;
    QString customerName;
    QString telephoneNumber;
    int number;
    int weight[5];
    QDate date;
};
```

Regarding the actual turkeys on the farm, they shall be assigned in another **struct**; this time, defining the order number they have been assigned to, their unique number and their weights (to the nearest 5g). As the order numbering

starts at the number 1, the number 0 for the order number shall be reserved for *unassigned*.

```
struct TurkeyData {
    int uniqueNumber;
    int weight;
    int order;
};
```

The **QValueVector** arrays are defined in the **private:** section of the class declaration, so that they are global to the class, but accessible only by members of the class.

```
typedef QValueVector<OrderData*> OrderArray;
typedef QValueVector<TurkeyData*> TurkeyArray;

OrderArray *m_orders;
TurkeyArray *m_turkeys;
```

In order to store the main data (not the order data) for the program, an XML configuration system will be used. This will be based upon KDE's **KConfig XT** system, a configuration framework based on XML.

Entities need to be defined within the **KConfig XT** XML configuration file to state which keys need to be stored in the main configuration file.

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE kcfg SYSTEM "http://www.kde.org/standards/kcfg/1.0/kcfg.
dtd">
<kcfg>
  <kcfgfile name="turkeyrc"/>
  <group name="turkey">
    <entry name="TurkeyCount" type="Int">
      <label>How many turkeys are in the farm</label>
      <default>0</default>
    </entry>
    <entry name="TurkeyOrderCount" type="Int">
      <label>How many turkeys have been ordered</label>
      <default>0</default>
    </entry>
  </group>
</kcfg>
```

This sets a key addressable by the name 'TurkeyCount' and sets its default value to 0. As turkeys are added to the database, this will be incremented.

Upon loading, the number of turkeys ordered and in total are loaded into the program's memory to be shown in the labels on the main window.

```

m_mainWindow->m_turkeysLeft->display(Settings::turkeyCount() -
    Settings::turkeyOrderCount());
m_mainWindow->m_turkeysOrdered->display(
    Settings::turkeyOrderCount());

```

When saving the data, the following functions are defined to automatically write to the files 'turkeys.dat' and 'orders.dat' in the current directory the data for the farm's turkeys and orders.

For the turkeys:

```

QFile file("turkeys.dat");

if (file.open(IO_WriteOnly)) {
    QTextStream stream(&file);

    QString temp;
    TurkeyArray::iterator it;

    for(it = m_turkeys->begin(); it != m_turkeys->end(); ++it) {
        stream << (*it)->uniqueNumber << endl;
        stream << (*it)->weight << endl;
        stream << (*it)->order << endl;
    }
}
file.close();

```

And for the orders:

```

QFile file("orders.dat");

if (file.open(IO_WriteOnly)) {
    QTextStream stream(&file);

    QString temp;
    int i = 1;
    OrderArray::iterator it;

    for(it = m_orders->begin(); it != m_orders->end(); ++it) {
        stream << (*it)->orderNumber << endl;
        stream << (*it)->customerName << endl;
        stream << (*it)->telephoneNumber << endl;
        stream << (*it)->number << endl;

        for (i=0; i<(*it)->number; ++i) {
            stream << (*it)->weight[i] << endl;
        }
    }
}

```

```

        stream << (*it)->date.year() << endl;
        stream << (*it)->date.month() << endl;
        stream << (*it)->date.day() << endl;
    }
}
file.close();

```

In order to load the data upon startup, two more functions are defined.  
For the turkeys:

```

QFile file("turkeys.dat");

if (file.open(IO_ReadOnly)) {
    QTextStream stream(&file);

    QString temp;

    while (!stream.atEnd()) {
        m_turkeys->push_back(new TurkeyData);

        temp = stream.readLine();
        m_turkeys->last()->uniqueNumber = temp.toInt();

        temp = stream.readLine();
        m_turkeys->last()->weight = temp.toInt();

        temp = stream.readLine();
        m_turkeys->last()->order = temp.toInt();
    }
}

if (m_turkeys->count() == 0)
m_turkeys->push_back(new TurkeyData);
file.close();

```

And for the orders:

```

QFile file("orders.dat");

if (file.open(IO_ReadOnly)) {
    QTextStream stream(&file);

    QString temp;
    int it, ii, y, m, d;

    while (!stream.atEnd()) {

```

```

m_orders->push_back(new OrderData);

temp = stream.readLine();
m_orders->last()->orderNumber = temp.toInt();

temp = stream.readLine();
m_orders->last()->customerName = temp;

temp = stream.readLine();
m_orders->last()->telephoneNumber = temp;

temp = stream.readLine();
it = temp.toInt();

m_orders->last()->number = it;

for (ii=0; ii < it; ++ii) {
    temp = stream.readLine();
    m_orders->last()->weight[ii] = temp.toInt();
}

temp = stream.readLine();
y = temp.toInt();

temp = stream.readLine();
m = temp.toInt();

temp = stream.readLine();
d = temp.toInt();

m_orders->last()->date.setYMD(y, m, d);
}
}
file.close();

if (m_orders->count() == 0)
    m_orders->push_back(new OrderData);

```

In order to print a bill for a customer, the following code is used. It brings up a dialogue to specify which printer to print to, and then prints out the bill.

```

OrderArray::iterator it;
QString temp;
int i;

QPrinter *printer = new QPrinter;

```

```

printer->setPageSize(QPrinter::A4);

if(printer->setup())
{
    QPainter * painter = new QPainter(printer);
    painter->setPen(QPen(QColor(Qt::black)));
    temp.setNum((*it)->orderNumber);

    painter->drawText(80, 20, "TERRY'S TURKEYS");
    painter->drawText(80, 80, "Bill Number: " +
        m_createOrderWindow->m_orderNumber->text());
    painter->drawText(80, 100, "Customer Name: " +
        m_createOrderWindow->m_customerName->text());
    painter->drawText(80, 120, "Telephone Number: " +
        m_createOrderWindow->m_telephoneNumber->text());

    temp.setNum(m_createOrderWindow->m_turkeyView->childCount());
    painter->drawText(80, 140, "Number of Turkeys Ordered: " +
        temp);
    painter->drawText(80, 180, "Weights/Prices of Turkeys Ordered
        :-");

    for (i = 0; i < m_createOrderWindow->m_turkeyView->
        childCount(); ++i) {
        painter->drawText(80, 200 + (20*i), m_createOrderWindow->
            m_turkeyView->itemAtIndex(i)->text(2) + "g - " +
            m_createOrderWindow->m_turkeyView->itemAtIndex(i)->
            text(1));
    }

    painter->drawText(80, 220 + (20*i), "Discount: " +
        m_createOrderWindow->m_discount->text());
    painter->drawText(80, 260 + (20*i), "Preparation Surcharge: " +
        m_createOrderWindow->m_surcharge->text());
    painter->drawText(80, 300 + (20*i), "Total Cost: " +
        m_createOrderWindow->m_totalCost->text());
    painter->end();
}

```

### 3 Implementation

Upon loading of the 'Create New Order' window, the following code is executed in order to set the date to the current date. Also, in order to ensure only certain characters are entered into the line edit for the telephone, a **KRestrictedLine** will be used. This is a class which inherits **KLineEdit** and the characters which

are allowed to be entered can be defined. The order number of the order is also defined from the class-wide integer defining how many orders have been placed thus far.

```
m_createOrderWindow->m_orderNumber->setNum(m_orderCount);
m_createOrderWindow->m_telephoneNumber->setValidChars
    ("1234567890+() ");
m_createOrderWindow->m_dateSelect->setDate(QDate::currentDate());
```

To ensure that no more than five turkeys per order are made, the function which adds the turkeys to the list checks that the number of turkeys already in the list does not exceed five. If it does, the function refuses to allow the user to add the turkey and informs the user that there are already five turkeys on the order. It also checks whether there are enough turkeys in the farm for the order. Likewise, if not, the user is informed.

```
QString temp, temp2;

if (m_createOrderWindow->m_turkeyView->childCount() + 1 >
    (Settings::turkeyCount() - Settings::turkeyOrderCount())) {
    KMessageBox::error(this, "Not enough turkeys in farm to add
        turkey to this order.", "Error");
    return;
}

if (m_createOrderWindow->m_turkeyView->childCount() < 5) {
    temp2.setNum(4.78 * m_createOrderWindow->m_weight->value());

    temp = temp2.right(2);
    if (temp.contains('.') )
        temp2.insert(temp2.length(), '0');

    temp.setNum(m_createOrderWindow->m_weight->value());

    m_createOrderWindow->m_turkeyView->insertItem(new
        KListViewItem(m_createOrderWindow->m_turkeyView, temp,
            temp2));

    m_mainWindow->m_turkeysOrdered->display(Settings::
        turkeyOrderCount());

} else {
    KMessageBox::error(this, "Can not add more than five
        turkeys to an order.", "Error");
}
```

For the preparation surcharge, discount and total cost labels, the following code is executed when the user confirms the order. In order to ensure the

formatting of the currency values is correct (i.e. - to 2 decimal places with a pound symbol in front), the following code is used to check if the decimal point is visible in the last two characters of the string and if so, to pad out the right hand side with a '0' accordingly. There is also a conditional statement to check whether the requested turkey weight and the actual assigned turkey weight differ by more than 1kg. If they do, then it calculates the discount accordingly.

```
QString temp, temp2, temp3;

float total = 0;
int i = 0;
float requestedWeight, actualWeight, discount = 0;

for (i=0; i < m_createOrderWindow->m_turkeyView->childCount();
    ++i) {
    temp = m_createOrderWindow->m_turkeyView->itemAtIndex(i)->
        text(0);
    requestedWeight = temp.toFloat() * 1000;

    temp = m_createOrderWindow->m_turkeyView->itemAtIndex(i)->
        text(2);
    actualWeight = temp.toFloat();

    if ((requestedWeight - actualWeight > 1000 ||
        actualWeight - requestedWeight > 1000) &&
        actualWeight != 0) {
        discount += (0.05 * 4.78) * (requestedWeight / 1000);
    }

    temp = m_createOrderWindow->m_turkeyView->itemAtIndex(i)->
        text(1);
    total += temp.toFloat();
}

total -= discount;
total += 2.5 * (m_createOrderWindow->m_turkeyView->childCount());

temp.setNum(total);
temp2 = temp.right(2);

if (temp2.contains('.') )
    temp.insert(temp.length(), '0');

temp.insert(0, '');

m_createOrderWindow->m_totalCost->setText(temp);
```

```

total = 2.5 * (m_createOrderWindow->m_turkeyView->childCount());

temp.setNum(total);
temp2 = temp.right(2);

if (temp2.contains('.'))
    temp.insert(temp.length(), '0');

temp.insert(0, '');

m_createOrderWindow->m_surcharge->setText(temp);

temp.setNum(discount);
temp2 = temp.right(2);

if (temp2.contains('.'))
    temp.insert(temp.length(), '0');

temp.insert(0, '');

m_createOrderWindow->m_discount->setText(temp);

```

When the 'Delete Turkey' button is pressed, the turkey currently highlighted in the ListView is deleted:

```

delete m_createOrderWindow->m_turkeyView->
    selectedItems(true).first();

```

When the confirm button is pressed, it checks whether all the input fields have been filled, as well as ensuring the date is in the correct region (between November 1<sup>st</sup> and December 24<sup>th</sup>). If everything is fine, then it adds the order to the **QValueVector** and closes the dialogue. If not, an error message opens.

```

if (m_createOrderWindow->m_customerName->text().length() == 0 ||
    m_createOrderWindow->m_telephoneNumber->text().length() == 0 ||
    m_createOrderWindow->m_turkeyView->childCount() == 0) {
    KMessageBox::error(this, "Please fill in all the information
        before confirming.", "Error");
    return;
}

if (m_createOrderWindow->m_dateSelect->date().month() < 11 ||
    (m_createOrderWindow->m_dateSelect->date().month() == 12 &&
    m_createOrderWindow->m_dateSelect->date().day() > 24)) {
    KMessageBox::error(this, "Invalid date; must be between
        November 1st and December 24th.", "Error");
}

```

```

        return;
    }

    OrderArray::iterator it = m_orders->begin();
    if ((*it)->orderNumber == 0) {
        m_orders->erase(it);
    }

    m_orders->push_back(new OrderData);

    m_orders->last()->orderNumber = m_orderCount;
    m_orders->last()->customerName = m_createOrderWindow->
        m_customerName->text();
    m_orders->last()->telephoneNumber = m_createOrderWindow->
        m_telephoneNumber->text();
    m_orders->last()->number = m_createOrderWindow->
        m_turkeyView->childCount();

    int i;
    QString temp;

    for (i=0; i < m_createOrderWindow->m_turkeyView->
        childCount(); ++i) {
        temp = m_createOrderWindow->m_turkeyView->itemAtIndex(i)->
            text(0);
        m_orders->last()->weight[i] = int(temp.toFloat() * 1000);
    }

    m_orders->last()->date = m_createOrderWindow->m_dateSelect->
        date();

    m_orderCount++;

    Settings::setTurkeyOrderCount(Settings::turkeyOrderCount() +
        m_createOrderWindow->m_turkeyView->childCount());
    Settings::writeConfig();

    m_mainWindow->m_turkeysOrdered->display(
        Settings::turkeyOrderCount());
    m_mainWindow->m_turkeysLeft->display(
        Settings::turkeyCount() -
        Settings::turkeyOrderCount());

    m_createOrderWindow->close();
    saveOrderData();

```

Upon clicking the ‘Delete Order’ push button, the following code is executed to remove the order from the **QValueVector** and then to resync with the database:

```
OrderArray::iterator it;

for (it = m_orders->begin(); it != m_orders->end(); ++it) {
    if (m_createOrderWindow->m_orderNumber->text().toInt() ==
        (*it)->orderNumber) {
        m_orders->erase(it);
        break;
    }
}

saveOrderData();
```

When the user clicks the ‘List All Orders’ pushbutton, the following code is executed to load the dialogue. This is so that the orders made previously are shown in the list view on the dialogue window.

```
m_allOrdersWindow = new AllOrdersWindow(0);
m_allOrdersWindow->show();

QString temp, temp2;

OrderArray::iterator it;
for(it = m_orders->begin(); it != m_orders->end(); ++it) {
    temp.setNum((*it)->orderNumber);
    temp2.setNum((*it)->number);

    m_allOrdersWindow->m_orderList->insertItem(new
        KListViewItem(m_allOrdersWindow->m_orderList, temp,
            (*it)->customerName, (*it)->date.toString(Qt::ISODate),
            temp2));
}
```

When an entity is clicked on the list view, the user is brought to the ‘Create New Order’ screen again where the data for that order can be modified. Because turkeys may have been assigned to an order number already, the code sorts the list by increasing weight for both the requested and actual weights, then matches up the lowest requested weight with the lowest actual weight, and so on, until the turkeys are all matched up. It then checks to see if there are any two turkeys which have more than a 1kg difference in weight and applies a discount accordingly.

```
QString temp = item->text(0);
QString temp2;
```

```

m_createOrderWindow = new CreateOrderWindow(0);
m_createOrderWindow->show();

connect(m_createOrderWindow->m_addTurkey, SIGNAL(clicked()),
        this, SLOT(slotAddTurkey()));
connect(m_createOrderWindow->m_confirm, SIGNAL(clicked()),
        this, SLOT(slotConfirmOrder()));
connect(m_createOrderWindow->m_deleteTurkey, SIGNAL(clicked()),
        this, SLOT(slotDeleteTurkey()));
connect(m_createOrderWindow->m_print, SIGNAL(clicked()),
        this, SLOT(slotPrintBill()));
connect(m_createOrderWindow->m_delete, SIGNAL(clicked()),
        this, SLOT(slotDeleteOrder()));

OrderArray::iterator it;
float f;

for(it = m_orders->begin(); it != m_orders->end(); ++it) {
    if((*it)->orderNumber == temp.toInt()) {
        m_createOrderWindow->m_dateSelect->setDate((*it)->date);
        temp.setNum((*it)->orderNumber);
        m_createOrderWindow->m_orderNumber->setText(temp);
        m_createOrderWindow->m_customerName->setText(
            (*it)->customerName);
        m_createOrderWindow->m_telephoneNumber->setText(
            (*it)->telephoneNumber);

        for (int i = 0; i < (*it)->number; ++i) {
            f = (*it)->weight[i];

            temp2.setNum(4.78 * (f/1000));

            temp = temp2.right(2);
            if (temp.contains('.') )
                temp2.insert(temp2.length(), '0');

            temp.setNum(f / 1000);

            m_createOrderWindow->m_turkeyView->insertItem(new
                KListViewItem(m_createOrderWindow->m_turkeyView,
                    temp, temp2));
        }
    }
}

```

```

m_createOrderWindow->m_turkeyView->setSortOrder(Qt::Ascending);
m_createOrderWindow->m_turkeyView->setSortColumn(0);

TurkeyArray::iterator iit;
QString temp3;
QStringList strlist;

temp = item->text(0);

for(iit = m_turkeys->begin(); iit != m_turkeys->end(); ++iit) {
    if ((*iit)->order == temp.toInt()) {
        temp3.setNum((*iit)->weight);
        strlist += temp3;
    }
}

strlist.sort();

int i = 0;
for (QStringList::Iterator itt = strlist.begin(); itt !=
    strlist.end(); ++itt) {
    m_createOrderWindow->m_turkeyView->itemAtIndex(i)->setText(2,
        (*itt));
    i++;
}

updateSubtotal();

```

When pushing the ‘Assign Turkeys’ button, the ‘Assign Turkeys’ dialogue is brought up and some code executed to restrict the **KRestrictedLine** to only allowing numeric input. It then adds the turkeys currently in the database to the list view so that they can be seen:

```

m_assignTurkeysWindow = new AssignTurkeysWindow(0);
m_assignTurkeysWindow->show();
m_assignTurkeysWindow->m_uniqueNumber->setValidChars("1234567890");

TurkeyArray::iterator it;
QString temp, temp2, temp3;

for(it = m_turkeys->begin(); it != m_turkeys->end(); ++it) {
    temp.setNum((*it)->uniqueNumber);
    temp2.setNum((*it)->weight);

    if ((*it)->order == 0)
        temp3 = "No";
}

```

```

else
    temp3.setNum((*it)->order);

m_assignTurkeysWindow->m_turkey->insertItem(new
    KListViewItem(m_assignTurkeysWindow->m_turkey,
        temp, temp2, temp3));
}

```

In this dialogue, when pressing the ‘Add Turkey’ button, the system checks that all the required fields are filled in and then prompts the user with an input dialogue to ask for which order number to add the turkey to. If the order number does not exist, it will inform the user. If the order does exist, it sets the relevant field in the **struct** to the order number and then saves the data.

```

QString temp;

int assign = QDialog::getInteger("Assign Turkey",
    "Order number to assign turkey to:");

temp.setNum(assign);

int ii = 0;

for (int i = 0; i < m_assignTurkeysWindow->
    m_turkey->childCount(); ++i) {
    if (m_assignTurkeysWindow->m_uniqueNumber->text() ==
        m_assignTurkeysWindow->m_turkey->itemAtIndex(i)->
        text(0)) {
        ii = 1;
    }
}

OrderArray::iterator it;
for(it = m_orders->begin(); it != m_orders->end(); ++it) {
    if (temp.toInt() == (*it)->orderNumber) {
        ii = 1;
    }
}

if (ii != 1) {
    QMessageBox::error(this, "Order number does not exist.",
        "Error");
    return;
}

m_assignTurkeysWindow->m_turkey->selectedItems(true).first()->

```

```

    setText(2, temp);

TurkeyArray::iterator iit;
for(iit = m_turkeys->begin(); iit != m_turkeys->end(); ++iit) {
    if (m_assignTurkeysWindow->m_turkey->selectedItems(true).
        first()->text(0).toInt() == (*iit)->uniqueNumber) {
        (*iit)->order = temp.toInt();
    }
}

saveTurkeyData();

```

## 4 Entity Relationship Diagram

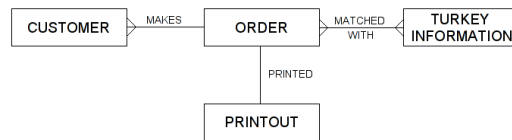


Figure 4: Entity Relationship Diagram

## 5 Security and Integrity of Data

### 5.1 Security

As the application is for a small business, no internal security in the application is needed as physical security is much better as an option than any sort of software security. However, care should be taken to make sure that the computer itself is secure to an extent. The following should be adhered to:

- The computer should have a power-on boot password
- The computer should be switched off when not in use
- The computer should be kept in a locked room

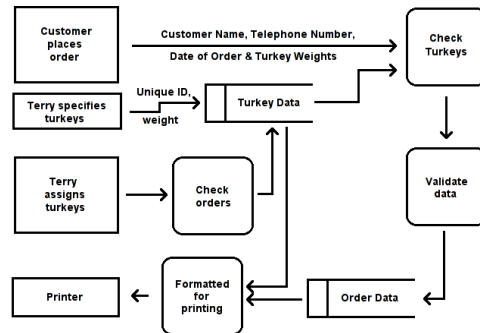
### 5.2 Data Integrity

The data files should be backed up regularly. A high-redundancy backup storage system such as a RAID array or tape backup would be far too expensive for a small farm such as this; backing up to a floppy diskette should be sufficient.

The data file shall be written every time a change is made to the data; this is so that if there is an unexpected power failure, the data will already have been written to disk and the chances of data loss are low.

## 6 System Design

Data flow diagram showing the design of the program around handling the data:



## 7 Testing

### 7.1 Test Plan

In order to test the application, I will have two data sets; the data set for the turkeys and the data set for the orders. There will be twelve orders and forty turkeys defined in the data sets.

I will then systematically add the orders to the database and add the turkeys into their database. I will then match up orders such that there will be some where the requested weights differ greatly from the actual weights, and some with no significant difference in weight. This will test the discounting algorithm. I will check the calculations outputted by the computer manually. I will also simulate errors which require the order to be cancelled then restarted to ensure the application can handle such scenarios.

For each order the following details will be checked:

- The unique order number is unique
- The number of unordered turkeys and ordered turkeys stays correct
- That no more than 5 turkeys can be ordered in each order
- That the date is validated properly
- That the number of turkeys ordered does not exceed the number of turkeys in the farm

Three orders will then be chosen and turkeys assigned to them. They will be assigned in such a way that one of the orders will be eligible for a discount because of an overweight turkey, one of the orders will be eligible for a discount because of an underweight turkey, and the other order will not be eligible for discount at all.

## 7.2 Test Data

Name: George Wright  
Telephone Number: 020 8987 2535  
Turkeys to be ordered:  $2 \times 5.5\text{kg}$   
Date of Order: 24th November 2004

Name: Arnold Rimmer  
Telephone Number: +76 (0)56 817 5424  
Turkeys to be ordered:  $1 \times 6.5\text{kg}$ ,  $3 \times 5.0\text{kg}$   
Date of Order: 11th December 2004

Name: David Lister  
Telephone Number: 07738 173 483  
Turkeys to be ordered:  $1 \times 6.0\text{kg}$ ,  $1 \times 7.5\text{kg}$   
Date of Order: 15th October 2004  
Note: To check date validation

Name: Kryten  
Telephone Number: 0183 498 3144  
Turkeys to be ordered:  $1 \times 5.5\text{kg}$ ,  $2 \times 8.5\text{kg}$ ,  $3 \times 9.5\text{kg}$   
Date of Order: 13th November 2004  
Note: To check maximum limit of 5 turkeys

Name: Duane Dibbley  
Telephone Number: 058 8176 5235  
Turkeys to be ordered:  $4 \times 5.5\text{kg}$ ,  $1 \times 10.5\text{kg}$   
Date of Order: 19th December 2004  
Note: To check maximum limit of 10kg

Name: Holly  
Telephone Number: +86 (0)24 174 4532  
Turkeys to be ordered:  $1 \times 7.5\text{kg}$   
Date of Order: 25th December 2004  
Note: To check upper date limit

Name: Charles Samuels  
Telephone Number: 020 2544 5565  
Turkeys to be ordered:  $5 \times 8.0\text{kg}$   
Date of Order: 14th November 2004

Name: Jonathan Riddell  
Telephone Number: 0113 9128 483  
Turkeys to be ordered:  $2 \times 5.5\text{kg}$ ,  $2 \times 6.0\text{kg}$   
Date of Order: 19th November 2004

Name: Jono Bacon  
Telephone Number: 0173 4814 8234  
Turkeys to be ordered: 4 × 9.5kg  
Date of Order: 23rd December 2004

Name: Jeff Snyder  
Telephone Number: 0134 9245 245  
Turkeys to be ordered: 1 × 5.5kg, 3 × 8.5kg  
Date of Order: 28th November 2004

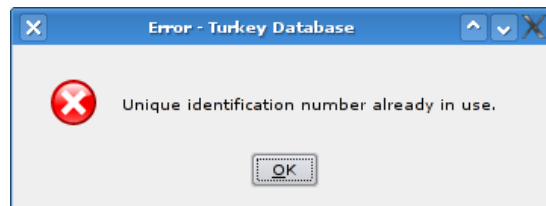
Name: Chris Howells  
Telephone Number: 0217 9172 471  
Turkeys to be ordered: 2 × 7.5kg  
Date of Order: 23rd November 2004

Name: Jim Conner  
Telephone Number: 07678 178 234  
Turkeys to be ordered: 4 × 5.5kg  
Date of Order: 10th December 2004

For the turkeys, forty random weights shall be chosen between 5kg and 10kg. Their unique numbers shall be numbered from 1000 to 1039.

### 7.3 Test Results

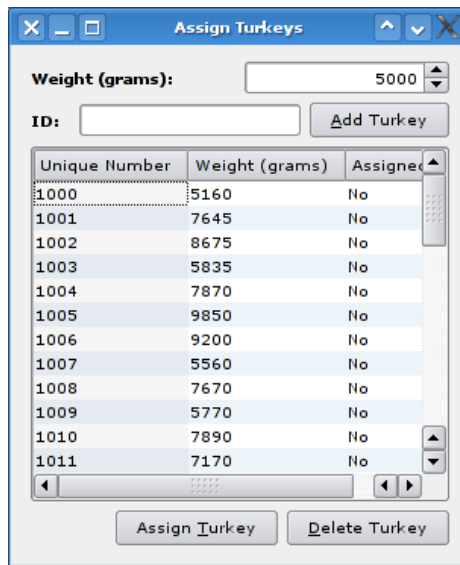
When adding the turkey data, if the user were to accidentally try to add a turkey with a unique identification which is the same as one already defined, the application would prompt the following error and refuse to add the turkey:



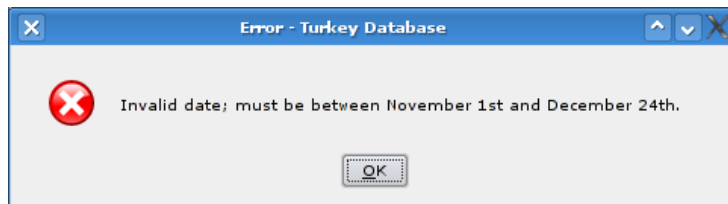
The counter on the main window worked properly when adding turkeys to the database and reached a total of 40.



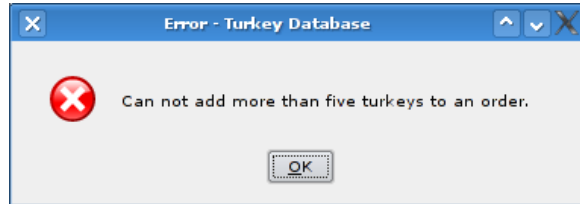
When all the turkeys were added, they were listed properly in the list view:



When trying to add an order with the wrong date, the following error message came up and the order was not confirmed:



When trying to add more than 5 turkeys to an order, the following error message come up and the order was not confirmed:

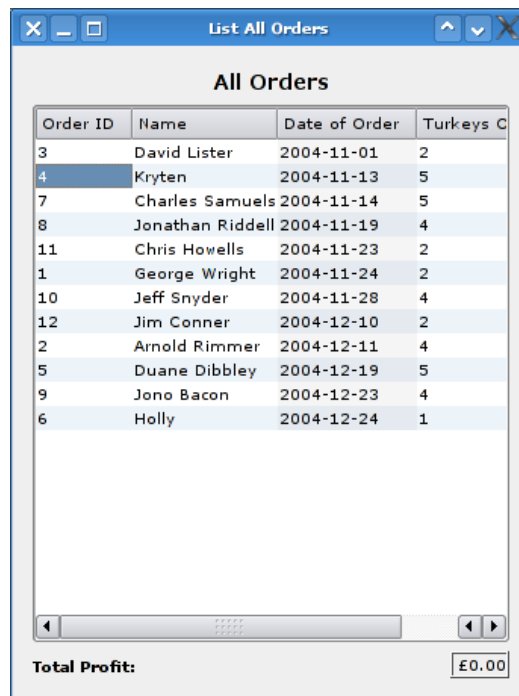


When trying to add an order for the 25th December, the same error message popped up as for the invalid date.

On attempting to add Jim Conner's last two turkeys, the following error message came up because the 40 turkey limit had already been reached:



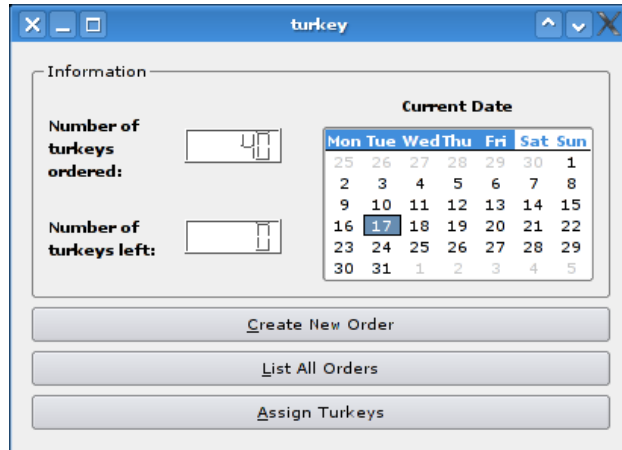
And after having added all the data, the order listing window looked like this (sorted by date):



A window titled "List All Orders" containing a table of orders. The table has columns for Order ID, Name, Date of Order, and Turkeys C. The orders are sorted by date. At the bottom, there is a "Total Profit:" label and a text box containing "£0.00".

Order ID	Name	Date of Order	Turkeys C
3	David Lister	2004-11-01	2
4	Kryten	2004-11-13	5
7	Charles Samuels	2004-11-14	5
8	Jonathan Riddell	2004-11-19	4
11	Chris Howells	2004-11-23	2
1	George Wright	2004-11-24	2
10	Jeff Snyder	2004-11-28	4
12	Jim Conner	2004-12-10	2
2	Arnold Rimmer	2004-12-11	4
5	Duane Dibbley	2004-12-19	5
9	Jono Bacon	2004-12-23	4
6	Holly	2004-12-24	1

And the main window thus:



## 7.4 Data Files

The data file `turkeys.dat` after adding all the turkeys looked as follows, where the first number is the turkey number; the second the weight and the third the order number it has been assigned to.

```
1000
5160
0
1001
7645
0
1002
8675
0
1003
5853
0
1004
7870
0
1005
9850
0
1006
9200
0
1007
5560
```

0  
1008  
7670  
0  
1009  
5770  
0  
1010  
7890  
0  
1011  
7170  
0  
1012  
5655  
0  
1013  
6560  
0  
1014  
7680  
0  
1015  
9570  
0  
1016  
10000  
0  
1017  
6560  
0  
1018  
7685  
0  
1019  
5255  
0  
1020  
7625  
0  
1021  
8655  
0  
1022  
8250  
0

1023  
9250  
0  
1024  
9755  
0  
1025  
6560  
0  
1026  
5015  
0  
1027  
7890  
0  
1028  
8675  
0  
1029  
6455  
0  
1030  
7565  
0  
1031  
5565  
0  
1032  
9845  
0  
1033  
8125  
0  
1034  
8985  
0  
1035  
5785  
0  
1036  
7635  
0  
1037  
8725  
0  
1038

9520  
0  
1039  
9745  
0

As for the order data, `orders.dat`, the data file is as follows with the first key being the order number; the second the customer name; the third the telephone number; the fourth the number of turkeys ordered; the next few (depending on how many turkeys were ordered) are the weights of the turkeys; the next three are the year, month and day of the order.

1  
George Wright  
020 8987 2535  
2  
5500  
5500  
2004  
11  
24  
2  
Arnold Rimmer  
+76 (0)56 817 5424  
4  
5000  
5000  
5000  
6500  
2004  
12  
11  
3  
David Lister  
07738 173 483  
2  
6000  
7500  
2004  
11  
1  
4  
Kryten  
0183 498 3144  
5  
5500

8500  
8500  
9500  
9500  
2004  
11  
13  
5  
Duane Dibbley  
058 8176 5235  
5  
10000  
5500  
5500  
5500  
5500  
2004  
12  
19  
6  
Holly  
+86 (0)24 174 4532  
1  
7500  
2004  
12  
24  
7  
Charles Samuels  
020 2544 5565  
5  
8000  
8000  
8000  
8000  
8000  
2004  
11  
14  
8  
Jonathan Riddell  
0113 9128 483  
4  
5500  
5500  
6000

6000  
2004  
11  
19  
9  
Jono Bacon  
0173 4814 8234  
4  
9500  
9500  
9500  
9500  
2004  
12  
23  
10  
Jeff Snyder  
0134 9245 245  
4  
5500  
8500  
8500  
8500  
2004  
11  
28  
11  
Chris Howells  
0217 9172 471  
2  
7500  
7500  
2004  
11  
23  
12  
Jim Conner  
07678 178 234  
2  
5500  
5500  
2004  
12  
10

## 7.5 Assigning Turkeys

Two orders have been considered; order number 1 and order number 2.

Regarding order number 1, a turkey which was over 1kg heavier than requested was assigned, and the other one was the correct weight for the other turkey order. In this, the discount was correctly calculated at being 1.31 pounds, and the total cost calculated at 56.27 pounds.

**Create New Order** 1

**Date:** 24 November 2004

**Customer Name:** George Wright

**Contact Telephone:** 020 8987 2535

**Weight:** 5.0

Weight (kg)	Price (£)	Assigned (grams)
5.5	26.29	5160
5.5	26.29	7645

**Turkey Preparation Surcharge:** £5

**Discount:** £1.3145

**Total Cost:** £56.2655

Upon pressing the 'Print' button, a dialogue opened asking which printer to print the document to. As the printout is to be on headed paper, the farm details are not printed. The printed output is as follows:

**TERRY'S TURKEYS**

**Bill Number: 1**

**Customer Name: George Wright**

**Telephone Number: 020 8987 2535**

**Number of Turkeys Ordered: 2**

**Weights/Prices of Turkeys Ordered :-**

**5160g - £26.29**

**7645g - £26.29**

**Discount: £1.3145**

**Preparation Surcharge: £5**

**Total Cost: £56.2655**

As for order number 2, all the turkeys assigned were within the 1kg range which does not allow for a discount. The application correctly detected this and did not assign a discount accordingly.

Create New Order <2>

**Create New Order**

**Date:**

**Customer Name:**

**Contact Telephone:**

**Weight:**

Weight (kg)	Price (£)	Assigned (grams)
5	23.90	5015
5	23.90	5560
5	23.90	5565
6.5	31.07	6560

**Turkey Preparation Surcharge:**

**Discount:**

**Total Cost:**

For the printing, the bill printout was fine:

TERRY'S TURKEYS

Bill Number: 2

Customer Name: Arnold Rimmer

Telephone Number: +76 (0)56 817 5424

Number of Turkeys Ordered: 4

Weights/Prices of Turkeys Ordered :-

5015g - £23.90

5560g - £23.90

5565g - £23.90

6560g - £31.07

Discount: £0

Preparation Surcharge: £10

Total Cost: £112.77

## 8 Bibliography

### **The KDE API documentation**

<http://developer.kde.org/documentation/library/cvs-api/>

### **The Qt API documentation**

<http://doc.trolltech.com/3.3/index.html>

### **The C Programming Language**

Brian W. Kernighan

Dennis M. Ritchie

### **The C++ Programming Language**

Bjarne Stroustrup

## **‘A’ Level Computing**

P. M. Heathcote

## **Understanding Computer Science for Advanced Level**

Ray Bradley

## **9 Miscellaneous Notes**

Document typeset in L<sup>A</sup>T<sub>E</sub>X.