**3 ФУНКЦИОНАЛЬНОЕ ПРОЕКТИРОВАНИЕ**

В этом разделе описывается функционирование и структура разрабатываемого программного обеспечения

**3.1 Описание функционирования программы**

Работа данного программного обеспечения можно описать следующей цепочкой действий:

* Выбор действия (архивация или разархивация)
* Выбор файла, обработка его содержимого (составление бинарного дерева для архивации и чтение таблицы кодировок для разархивации)
* Запись в новый файл, причем архиватор читает исходный файл небольшими порциями (к примеру, по 256 байт), обрабатывает их на основе бинарного дерева, то есть сжимает их, и записывает обработанные входные данные в выходной буфер такого же размера. Так программа продолжает считывать новые порции исходных данных, пока выходной буфер не заполнится полностью, далее выходной буфер записывается в файл и очищается. Цикл повторяется до тех пор, пока весь исходный файл не будет переведен в свою сжатую форму (рисунок 3.1).

Разархиватор также создает новый файл, но, в отличии от архивации, теперь программа считывает сжатые данные, и, на основе таблицы кодировок байтов, преобразовывает сжатую запись в код байта (рисунок 3.2).

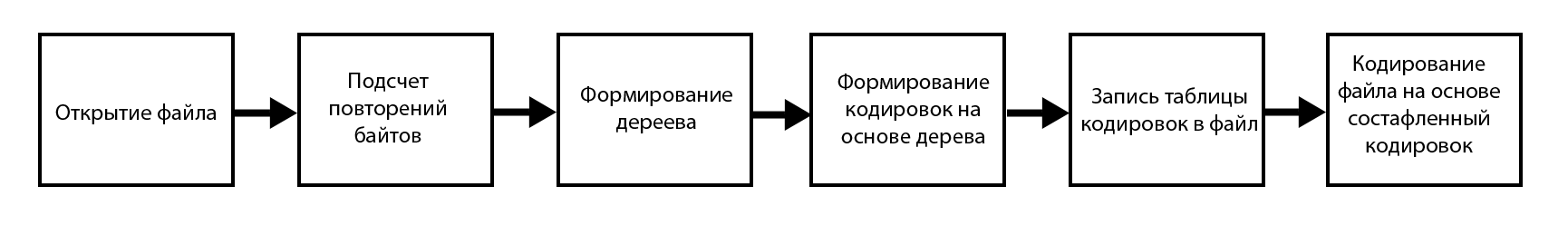


Рисунок 3.1 – Логика архивации файла

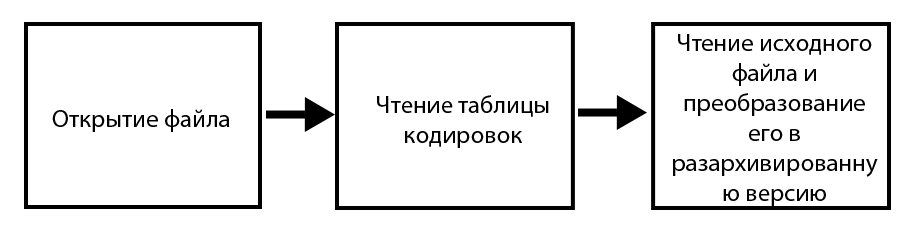


Рисунок 3.2 – Логика разархивации файла

**3.2 Описание структуры взаимодействия внутри приложения**

**3.2.1 Класс Node**

Таблица 3.1 – Шаблонный класс Node

|  |  |  |  |
| --- | --- | --- | --- |
| Private поля класса | | | |
| Имя | Тип | | Описание |
| has\_value | bool | | Необходимо для различия между листом и узлом дерева |
| Public поля класса | | | |
| Имя | Тип | | Описание |
| code | QString | | Содержит кодировку элемента в листе |
| count | T | | Содержит число, которое в листе значит число повторений байта в файле, а в узле является весом узла |
| value | N | | Значение узла дерева |
| left | Node<T, N>\* | | Указатель на левого наследника узла |
| right | Node<T, N>\* | | Указатель на правого наследника |
| Public методы класса | | | |
| Имя | Принимаемые параметры | Возвращаемое значение | Описание |
| Node | T cnt, N val | Node<T, N> | Конструктор новой ноды, устанавливает поле has\_value в true |
| Node | Node<T, N>\* lef, Node<T, N>\* rgt | Node<T, N> | Конструктор нового узла на основе двух существующих. Особенность в том, что has\_value = false, а значение поля count вычисляется как сумма аналогичных полей обоих наследников |
| ~Node | - | - | Вызывает delete для наследников, если они существуют |
| getCount | - | T | Возвращает значение поля count |

Продолжение таблицы 3.1

|  |  |  |  |
| --- | --- | --- | --- |
| Имя | Принимаемые параметры | Возвращаемое значение | Описание |
| getValue | - | N | Возвращает значение поля value |
| getCode | - | QString | Возвращает кодировку листа |
| hasValue | - | bool | Возвращает значение поля has\_value |
| operator> | Node<T, N> other | bool | Сравнивают два узла по значению поля count |
| operator< |
| operator== |
| operator!= |

**3.2.2 Класс NodeComparator**

Таблица 3.2 – Класс NodeComparator

|  |  |  |  |
| --- | --- | --- | --- |
| Public методы класса | | | |
| Имя | Принимаемые параметры | Возвращаемое значение | Описание |
| operator() | Node<int, QChar> \*first,  Node<int, QChar> \*second | bool | Компаратор, который используется в приоритетной очереди, для создания приоритета по значению (Впереди будут узлы с наименьшим значением count) |

**3.2.3 Класс bTree**

Таблица 3.3 – Класс bTree

|  |  |  |
| --- | --- | --- |
| Private поля класса | | |
| Имя | Тип | Описание |
| root | Node<int, QChar> \* | Главный узел дерева |
| dictionary | QMap<QChar, QString> | Словарь кодировок |

Продолжение таблицы 3.3

|  |  |  |  |
| --- | --- | --- | --- |
| Public методы класса | | | |
| Имя | Принимаемые параметры | Возвращаемое значение | Описание |
| bTree | Node<int, QChar> \*nd | bTree | Конструктор дерева из существующего узла |
| ~bTree | - | - | Вызывает рекурсивный метод destroyTree |
| destroyTree | Node<int, QChar> \*&node | - | рекурсивно удаляет все узлы дерева |
| formCodes | - | - | Вызывает метод formCodesRec |
| formCodesRec | Node<int, QChar>\* node,  QString tempCode | - | Проходит по листьям дерева и составляет для них коды ,которые записываются в поле dictionary |
| getDictionary | - | QMap<QChar, QString>& | Возвращает ссылку на заполненный словарь – поле dictionary |