

华中科技大学

软件工程课程设计

蔬菜种植信息管理系统

院 系：计算机科学与技术学院

专业班级：CS1601

姓 名：刘本嵩

学 号：U201614531

2018 年 11 月 29 日

Table of Content

1 问题定义及可行性分析.....	1
1.1 系统的问题定义.....	1
1.2 系统环境.....	1
1.3 系统的可行性分析.....	1
2.需求分析.....	3
1、需要处理的基础数据.....	3
(1) 蔬菜种类信息表.....	3
(2) 蔬菜基本信息表.....	4
(3) 菜农种植信息表.....	4
2、系统基本功能.....	5
(1). 数据维护.....	5
(2). 数据查询.....	5
(3). 数据统计.....	6
(4). 数据存取.....	7
3 概要设计与详细设计.....	7
2.1 系统整体设计.....	7
2.2、数据结构设计.....	8
1. 蔬菜种类.....	9
2. 蔬菜信息.....	9
3. job.....	9
2.3 详细设计.....	13
4 测试报告.....	22
5 项目管理.....	27
6 体会与建议.....	29

1 问题定义及可行性分析

1.1 系统的问题定义

通过设计一个有效的蔬菜种植信息管理系统,不仅可以节省大量的人力物力,减少投资和填写,登记过程中可能出现的错误,而且极大的提高了查询效率和更新效率,并且是蔬菜种植信息管理变得系统化和自动化。

蔬菜种植信息管理系统用于菜农管理其蔬菜种植的相关信息,主要包括蔬菜种类基本信息,蔬菜基本信息和种植计划基本信息。

本蔬菜种植信息管理系统模块是为了实现蔬菜种植信息的科学管理而设计的,通过本系统,可以有效的管理蔬菜种类基本信息、蔬菜基本信息、种植计划基本信息,并且具有信息的增加,查询,修改等功能,而且能快速统计蔬菜种植计划的相关数据。

1.2 系统环境

本系统可运行在 linux 及其变种,BSD 或 MacOS 或 Solaris 等 unix 变种, windows xp 7 8 10 或 ReactOS 等 NT 内核变种等操作系统。

开发者硬件使用 Intel(R) Core(TM) i5-4200H CPU @ 2.80GHz,16GiB 物理 +16GiB 交换共计 32GiB 内存, 2320GiB 本地硬盘, NVIDIA 950M 独立 GPU。软件使用 ArchLinux rolling (kernel 4.19.2)。程序将运用 c 语言完成整体开发。

1.3 系统的可行性分析

蔬菜种植信息管理系统需要有软件工程,数据库技术,相应软硬件支持,经济支持,人才要求等。

软件工程是指导计算机软件开发和维护的一门工程科学,用当前最好的管理

解释和方法,经济的开发出高质量的软件并能够有效的维护它。他从六十年代末开始发展到现在已经有半个世纪的历史,研究范围广泛,包括各种新技术方法、工具和管理各方面,是一个异常活跃的研究领域。到现在已经形成了一套系统规范的知识体系,严格遵循软件工程方法可以大大提高软件开发成功的几率,显著减少软件开发维护的问题,为系统的开发和维护提供指导。

数据库技术 从诞生到现在,在不到半个世纪的时间里,形成了坚实的理论基础、成熟的商业产品和广泛的应用领域,吸引越来越多的研究者加入。数据库的诞生和发展给计算机信息管理带来了一场巨大的革命。随着应用的扩展与深入,数据库的数量和规模越来越大,数据库的研究领域也已经大大地拓广和深化了。数据库是一个充满活力和创新精神的领域。现在的数据库技术既能进行数据的集中和共享,又能有效的保持数据的独立性和抽象性,非常适合进行数据的管理。而且随着面向对象数据库的出现,是数据库的设计更加人性化,能更好的符合用户的要求,为系统设计提供了基础。

蔬菜种植管理系统的人员要求:系统分析人员,开发人员,数据库管理人员,系统测试人员。采用 c 语言实现,依靠强大的数据库控件和数据库管理系统和其他语言相结合,两个月内开发出系统。现在有很多专门的机构用来培养计算机人才,各大高校也设置了很多的相应课程,现在的社会优秀的 IT 人才层出不穷,为系统的开发提供了强有力的智力支持。

2.需求分析

现有某菜农，在给定面积的菜地里种植各种蔬菜，每年种植的蔬菜种类不完全相同，各种的收成也不一样，请设计一个程序，帮助该菜农管理每年种植的蔬菜品种、蔬菜秧苗数量（株）、收获蔬菜的数量（斤）。

1、需要处理的基础数据

对该菜农种植的蔬菜信息进行管理，主要包括蔬菜种类信息，蔬菜基本信息，菜农种植信息表等三类信息。

(1) 蔬菜种类信息表

中文字段名	类型及长度	举例
分类编码	char	'1'~'5'
分类名称	char[8]	5个分类名称：根茎类、果菜类、瓜类、叶菜类、菌类

根茎类：白萝卜，胡萝卜，大葱，小葱，蒜，洋葱，莴笋，山药，马铃薯、红薯，等等

果菜类：菜椒，青椒，尖椒，甜椒，朝天椒，线椒，南瓜，丝瓜，扁豆等等

瓜类：西瓜，甜瓜，白瓜，黄瓜，苦瓜等等

叶菜类：大白菜，小白菜，生菜，菠菜，韭菜，芹菜，空心菜等等

菌类：木耳，银耳，平菇，草菇，金针菇，香菇，等等

(2) 蔬菜基本信息表

中文字段名	类型及长度	举例
蔬菜编号	int	自增长(顺序增加)
蔬菜名称	char[20]	“白萝卜”
分类码	char	'1' //表示根茎类蔬菜
营养成分	Char[20]	指蔬菜中含义的矿物质成分，比如 菠菜含有铁

辣椒含有胡萝卜素、钙、铁；扁豆含有胡萝卜素、维生素 C、钾、磷、铁、锌；西红柿含有钙、磷、铁、硼、锰、铜；萝卜含有蛋白质、葡萄糖、维生素 C；胡萝卜含有胡萝卜素；空心菜含有蛋白质、胡萝卜素、维生素 B1、B2、C；苦瓜含有维生素 B2、维生素 C、钙、铁、磷；香菇含有多糖、糖种酶。等等。

(3) 菜农种植信息表

中文字段名	类型及长度	举例
编号	Int	自增长
蔬菜编号	int	同蔬菜基本信息表中的蔬菜编号
种植面积	Int	2：表示 2 分地

收获重量	float	公斤
种植年份	char[5]	“2015” 2015 年

2、系统基本功能

本系统需要实现数据维护，数据查询和数据统计三个主要功能模块，另外根据情况添加辅助功能模块。下面给出了三个主要模块的功能需求，辅助功能模块根据各人的理解和分析自己设计。

(1). 数据维护

本模块实现蔬菜种类信息、蔬菜基本信息、蔬菜种植信息等三方面基本信息的数据维护功能，又分为三个子模块。

- 蔬菜种类信息维护：包括对蔬菜种类信息的录入、修改和删除等功能。
- 蔬菜基本信息维护：包括对蔬菜基本信息的录入、修改和删除等功能。
- 蔬菜种植信息维护：包括对蔬菜种植信息的录入、修改和删除等功能。

(2). 数据查询

本模块实现蔬菜种类信息(五个分类：根茎类、果菜类、瓜类、叶菜类、菌类)，蔬菜基本信息，蔬菜种植信息等三方面基本信息的数据查询功能，又分为三个子模块。

1) 蔬菜种类信息查询功能

以分类编码为条件来查找并显示满足条件的蔬菜分类信息。例如，查找并显示分类编码为'3'的蔬菜分类信息。

2) 蔬菜基本信息查询功能

- 蔬菜名称中文字符子串为条件查找，并显示蔬菜中包含指定子串的蔬菜基本信息。例如，查找并显示果菜类蔬菜名称中包含为“椒”的果菜类蔬菜基本信息。
 - 以分类码和营养成分为条件查找并显示满足条件的蔬菜基本信息。例如，查找并显示分类码为'4'（叶菜类）且营养成分中含有铁的所有蔬菜基本信息。

3) 蔬菜种植信息查询功能

- 蔬菜部分名称（模糊查找）和种植年份为条件查找并显示满足条件的所有蔬菜种植信息。例如，查找并显示蔬菜名称中含有椒的蔬菜且在“2015”年种植的所有蔬菜信息。
 - 以蔬菜名称为条件查找并显示满足条件的蔬菜种植信息。例如，查找并显示蔬菜名称为“菠菜”的所有年份种植的菠菜信息（重量），并按年份分别显示。

(3). 数据统计

本模块实现五个方面的数据统计功能,前四个功能需求已给出,第三个自行设计。

- 分别统计某年各类蔬菜（如叶菜类、根茎类等）种植总面积、收获总重量，按总重量降序排序后，输出分类名称、种植面积、收获总重量（按种类统计）。

- 以所输入的起止年份为条件(如 2015-2017),按蔬菜名称(如菠菜、黄瓜等)统计该三年内所种各种蔬菜总面积、收获总重量,按总重量降序排序后,输出蔬菜名称、分类名称、种植面积、总重量。
- 分别统计某种类蔬菜的已有的数量。
- 给定某个营养成分,程序自动判定含有该营养成分的蔬菜,并且显示输出所有含有该营养成分的蔬菜名称。
- 有关该菜农蔬菜种植信息的其他方面数据统计。

(4). 数据存取

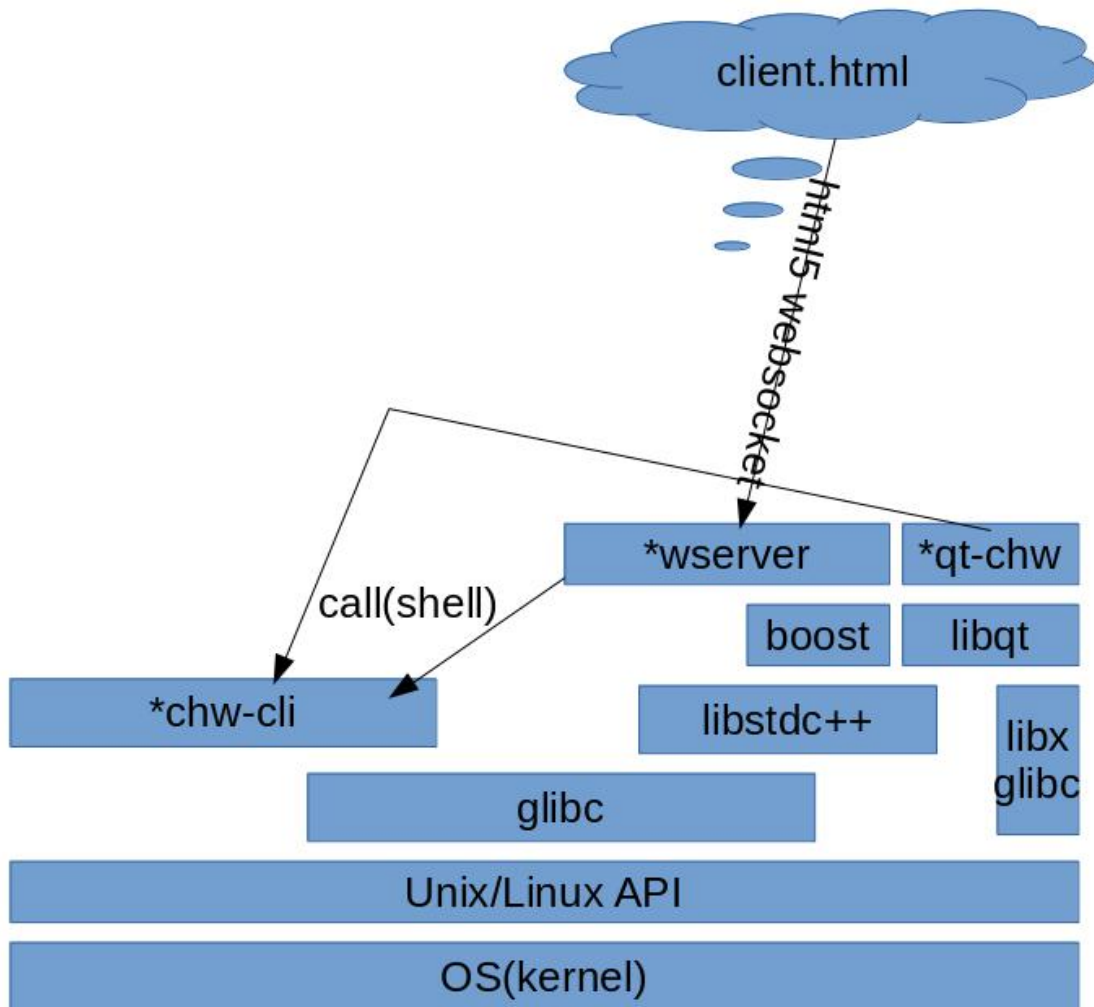
以上三种信息在程序运行时以链表结构形式存在,并且输的存储采用动态存储分配方式,同时在外存上以二进制文件的形式对数据进行存储,应保证数据在内存中与外存中内容的一致性。

上述四种功能是系统的基本功能,此外需根据情况增加辅助功能,使系统具有良好的人机交互界面,易于操作,并体现良好的健壮性和容错性。

3 概要设计与详细设计

总体设计是在功能需求分析的基础上,设计系统的功能结构和数据结构,即按照功能要求构造系统的功能框架,并确定数据的描述规范和数据之间的关系。通过总体设计建立目标系统的逻辑关系,系统功能框架通常用模块层次结构图来描述,能够直接地体现功能模块之间的调用关系。

2.1 系统整体设计



图中,在 released bin 中呈现的可执行文件为 chw-cli(Backend), wserver(websocket server to support client.html frontend), qt-chw(frontend implemented by libqt). chw-cli 通过 shell 传参给 main,通过 stdout(/dev/fd/1)输出信息,通过 stderr(/dev/fd/2)输出错误。shell 为一切进程间通信提供了高效而简洁的接口。专业的前端设计语言有利于加速开发和减少开发成本。模块间的低耦合性为代码的可维护性提供了根本保证。

此报告将认为代码的价值满足,可维护性>可扩展性>正确性>运行时间开销>运行内存开销>硬盘/其他外存储设备开销>编译开销。

2.2、数据结构设计

数据结构设计即确定数据的逻辑结构和物理结构,数据的逻辑结构是对数据之间关系的描述,数据的物理结构是数据逻辑结构在计算机中的表示。

按任务要求,系统需要处理的信息有三种:蔬菜种类(veg_class),蔬菜信息(veg_info),工作信息(job_info)。

另外,出于数据规范化、便于处理和节省存储空间等目的,我们还可以对信息中某些项进行编码,用固定长度的代码来表示长度不一的信息。

下面分别为本系统所涉及的代码数据、基础数据和生成数据的数据结构,以及数据在内存外存中的存储结构。

```
>>> ./src/app/data.h
typedef struct _veg_class {
    int code;
    char name[64];
} veg_class;

typedef struct _veg_info {
    int no; //Must self-inc autoly.
    char name[64];
    const veg_class *pclass;
    int __class_code; //Unused(!!Unmaintained!!). Just to make the
doc happy. //Defined as pclass->code
    char nutr_info[64];
} veg_info;

typedef struct _job_info {
    int no;
    const veg_info *pveg;
    int __veg_no; //Unused(!!Unmaintained!!). Just to make the doc
happy. //Defined as pveg->no
    int area;
    float weight;
    int year;
    char __year[5]; //Unused(!!Unmaintained!!). Just to make the doc
happy. //Defined as string(year)
} job_info;
<<< end data.h
```

1. 蔬菜种类

数据结构名称：蔬菜种类		数据结构标识：_veg_class			
数据项名称	数据项标识	数据类型	数据长度	取值范围	示例
代码	code	int	sizeof(int)	int	1
名字	name	CString	64	01-FF	"veg_cls_a"

2. 蔬菜信息

数据结构名称：蔬菜信息		数据结构标识：_veg_info			
数据项名称	数据项标识	数据类型	数据长度	取值范围	示例
序号	no	int	sizeof(int)		1
名称	name	CString	64		"vega"
种类	__class_c ode	int	sizeof(int)		1
营养	nutr_info	CString	64		"vc/Hg"

3. job

数据结构名称: job		数据结构标识: _job_info			
数据项名称	数据项标识	数据类型	数据长度	取值范围	示例
job 编号	no	int	sizeof(int)		1
蔬菜	__veg_no	int	sizeof(int)		2
种植面积	area	int	sizeof(int)		9999
种植质量	weight	float	sizeof(f)	0-	123.456
种植年份	__year	int	sizeof(int)	0-	1970

链表使用伪泛型实现, talk is cheap, I'll just show key code.

>>> job.c (Incomplete code)

```
#include "list.h"
rList jbuf;
jbuf = rList_newlist();
read_from_disk();
rList_for_each(iter, jbuf)
{
    job_info *p = (job_info *)iter->data;
    //Do some operation, print result to some pipe/fd.
}
write_to_disk();
rList_destructor(jbuf);
<<< end job.c
```

veg-class.c, veg.c 表现与 job.c 非常相似, 代码复用率高。这是使用抽象数据结构的好处。下面是链表库的设计, 高度复用了 glibc++ std::list 的源码。接口按标准风格封装, 命名和 std::list 保持严格一致。

>>> list.h

```
#ifndef SRC_LIBR_LIST
#define SRC_LIBR_LIST

struct rListNode{
    struct rListNode* prev;
    struct rListNode* next;
    void* data;
};

struct rListDescriptor{
    struct rListNode* first;
    struct rListNode* last;
};

struct rListNode* rList_constructor();
struct rListDescriptor* rList_newlist();
void rList_push_back(struct rListDescriptor* desc,void* data,unsigned
int length);
void rList_pop_back(struct rListDescriptor* desc);
void rList_push_front(struct rListDescriptor* desc,void* data,unsigned
```

```

int length);
void rList_pop_front(struct rListDescriptor* desc);
void rList_eraseElem(struct rListDescriptor* desc, struct rListNode*
target);
void rList_destructor(struct rListDescriptor* desc);
typedef struct rListDescriptor rListDescriptor;
typedef struct rListNode* rListIterator;
typedef rListDescriptor *rList;
#define rList_for_each(_var_iterator, _var_list) for(rListIterator
_var_iterator = _var_list->first; _var_iterator != nullptr;
_var_iterator = _var_iterator->next)
#endif
<<< end list.h

```

数据在外存中的存储由 librlist/fmtio.c 实现。这是简单的二进制 dump。

>>> fmtio.c

```

#include "fmtio.h"
#include <unistd.h>

```

```

int fmt_read_file(fd fin, rList dataBuf, size_t sizeofObj, const void
*delimiter)
// Pass nullptr as delimiter to use 0 bytes as delimiter. //Return read
obj counter, neg if error.
{
    bool mustFree = delimiter == nullptr;
    if(mustFree)
        delimiter = calloc(1, sizeofObj);
    int counter = 0;
    while(true)
    {
        void *mem = malloc(sizeofObj);
        auto bytes = read(fin, mem, sizeofObj);
        if(bytes == -1) die("read fin error. current counter=%d", counter);
        if(bytes < sizeofObj) break;
        if(0 == memcmp(mem, delimiter, sizeofObj)) break;
        rList_push_back(dataBuf, mem, sizeofObj);
        ++counter;
        free(mem);
    }
    if(mustFree)
        free((void *)delimiter);
    return counter;
}

```

```

int fmt_write_file(fd fout, rList dataBuf, size_t sizeofObj, const void

```

```

*delimiter)
// Pass nullptr as delimiter to use 0 bytes as delimiter. //Return read
obj counter, neg if error.
{
    bool mustFree = delimiter == nullptr;
    if(mustFree)
        delimiter = calloc(1, sizeofObj);
    int counter = 0;
    for(rListIterator it = dataBuf->first; it != nullptr; it = it->next)
    {
        auto ret = write(fout, it->data, sizeofObj);
        if(ret < sizeofObj) die("write fout error: write() return %d,
current counter=%d", ret, counter);
        ++counter;
    }
    auto ret = write(fout, delimiter, sizeofObj);
    if(ret != sizeofObj)
        die("write delimiter failed. write()=%d, counter=%d", ret,
counter);
    if(mustFree)
        free((void *)delimiter);
    return counter;
}
<<< fmtio.c

```

uthash 开源库使用纯宏实现, 不涉及 **struct**, 按规则不予列举。

2.3 详细设计

RealTime Status: master  release 

主要介绍 chw-cli。

首先,我必须介绍 project 的文件结构,以便进行整体结构的介绍。

建议直接在 <https://github.com/recolic/chw> 获得源文件。

recolic@RECOLICPC ~/CL/chw (master)> tree .

```
. #master 分支:没有对 c++代码进行预编译,打包和加密的纯代码分支。
├── .travis.yml #指导 travis ci 对每一次提交进行自动的在线编译,并自动更新文档中的 build status。
├── bin #预先在 Linux x86_64 下编译,可以直接部署的二进制文件。
│   ├── check.sh #编译时环境检查,验证程序要求的依赖是否满足。
│   ├── chw-cli #后端
│   ├── clear_database.sh #清空数据库。这个功能没有要求,因此单独实现
│   ├── gui #web gui files
│   │   ├── client.min.html #web gui here!
│   │   ├── favicon.ico
│   │   ├── form.min.css
│   │   └── sheet.min.js
│   ├── qt-gui #用 qt 重写的 gui
│   ├── README #说明
│   ├── runtime_check.sh #检查预先编译的程序是否能正常运行在这台主机
│   └── wserver #web socket server
├── LICENSE #开源许可证:修改过的 Mozilla 许可证,对抄袭课设的责任认定和 Fair use 的界定做了特别规定和修改。
├── README.md #编译说明文档 项目介绍文档
└── src #源文件
    ├── app #与逻辑有关的源文件
    │   ├── cmd.c #命令行参数解析器
    │   ├── cmd.h
    │   ├── data.h #定义数据结构
    │   ├── job.c #定义 job 有关的数据库操作
    │   ├── job.h
    │   ├── main.c #定义 main 函数,调用伪 c++类的构造函数和析构函数
    │   ├── shit-op.c #定义其他操作
    │   ├── shit-op.h
    │   ├── veg.c #定义蔬菜有关的数据库操作
    │   ├── veg-class.c #定义蔬菜种类有关的数据库操作
    │   ├── veg-class.h
    │   └── veg.h
    ├── cclib #Simple C Container Library. 已经停止维护并且 buggy.
    │   ├── bitstrings.c
    │   └── bloom.c
```

- | |— buffer.c
- | |— ccl_internal.h
- | ... 此处省略 cclib 源文件
- | |— wstrcollection.c
- | |— wstringlist.c
- | |— wstringlist.h
- |— check.sh #检查依赖和环境, 由 CMakeLists.txt 调用并进行系统自检
- |— clear_database.sh #清空数据库。这个功能没有要求, 因此单独实现
- |— cmake_clean.sh #清理 cmake 缓存
- |— CMakeLists.txt #CMake 主要配置文件, 指定了一切编译选项, 编译条件和编译方法, 用于自动生成 Makefile, 指导整个项目的编译与发布。
- |— common.h #定义了一些对 C 语言的常用扩展
- |— error.h #定义了错误处理和异常, 使用 php die 风格。
- |— librlist #自己实现的链表库
- | |— CMakeLists.txt #子编译配置文件, 指导 camke
- | |— fmtio.c #定义了内存中链表和外存相交换的方法
- | |— fmtio.h
- | |— list.c #定义了链表的基本操作
- | |— list.h
- | |— search.c #泛型链表算法, 未使用
- | |— search.h
- | |— sort.c #泛型链表算法, 未使用
- | |— sort.h
- |— libut #纯宏实现的 C Container Lib
- | |— LICENSE #libut 的开源许可证
- | |— utarray.h
- | |— uthash.h
- | |— utlist.h #未使用 utlist
- | |— utringbuffer.h
- | |— utstring.h
- |— mod
- | |— console.c #定义了更好的 system 函数, 与 shell 交互更便利。
- | |— console.h
- | |— filter.c #定义了类似 gnu grep 的过滤器, 为 ls 后的可选参数提供支持。
- | |— filter.h
- | |— _fmtio.c #已弃用 整合进 rList
- | |— _fmtio.h
- | |— fstr.c #动态的格式化字符串函数 C 语言版本已停止维护
- | |— fstr.h
- | |— hash.c #mod33 哈希函数
- | |— hash.h
- |— note.md
- |— qt #qt 部分与此段无关 不予介绍


```

|   |   |— build-untitled-unknown-Debug
|   |   |   |— inputer.o
|   |   |   |— main.o
|   |   |   |— mainwindow.o
|   |   |   |— Makefile
|   |   |   |— moc_inputer.cpp
|   |   |   |— moc_inputer.o
|   |   |   |— moc_mainwindow.cpp
|   |   |   |— moc_mainwindow.o
|   |   |   |— moc_predefs.h
|   |   |   |— ui_inputer.h
|   |   |   |— ui_mainwindow.h
|   |   |   |— untitled
|   |   |— inputer
|   |   |   |— inputer.cpp
|   |   |   |— inputer.h
|   |   |   |— inputer.ui
|   |   |— untitled
|   |   |   |— cmder.h
|   |   |   |— main.cpp
|   |   |   |— mainwindow.cpp
|   |   |   |— mainwindow.h
|   |   |   |— mainwindow.ui
|   |   |   |— untitled.pro
|   |   |   |— untitled.pro.user
|   |— runtime_check.sh #从源码编译的情形无需进行 runtime 检查
|   |— tests
|   |   |— a.out
|   |   |— libccl.a
|   |   |— test-console.c
|   |   |— test-ht.c
|   |   |— test-lib.c
|   |   |— test.sh #压力测试脚本 进行几万次的自动压力测试，保证可靠性
|   |— web #与 web 前端有关的组件
|   |   |— client # web 前端
|   |   |   |— client.html
|   |   |   |— favicon.ico
|   |   |   |— form.css
|   |   |   |— min
|   |   |   |   |— client.min.html
|   |   |   |   |— favicon.ico
|   |   |   |   |— form.min.css
|   |   |   |   |— sheet.min.js
|   |   |   |— sheet.js

```

```

└─ server #web socket 服务器
    └─ bin.h #C++到 C 的转换自动生成的文件
    └─ cmake_clean.sh #清理子目录的 cmake 缓存
    └─ CMakeLists.txt #指导此项目的编译 要求 C++14
    └─ libwebsocket++ #现代 C++实现的 websock 纯头库
        └─ websocketpp
            └─ base64
                └─ base64.hpp
            └─ client.hpp
            └─ CMakeLists.txt
            此处省略冗长无意义的列举
            └─ utilities.hpp
            └─ version.hpp
    └─ main.cc #main 函数所在地
    └─ make_trick.py #执行转换
    └─ mod
        └─ console.cc #修改 console.cc 使之适合 C++
        └─ console.h
    └─ rlib #完全自己实现的 C++库 不再进行说明
        └─ c
            └─ fake_cpp.h #向 C 语言中引入 C++类,vector 和简单的 tree.
        └─ macro.hpp
        └─ Makefile
        └─ noncopyable.hpp
        └─ nonmovable.hpp
        └─ print.hpp
        └─ README.md
        └─ require
            └─ cxx11
            └─ cxx14
            └─ cxx17
            └─ gcc
            └─ linux
            └─ win
        └─ scope_guard_buffer.hpp
        └─ scope_guard.hpp
        └─ string
            └─ fstr.hpp
            └─ string.hpp
        └─ sys
            └─ cc_codegen.py
            └─ cc_list
            └─ compiler_detector
            └─ fdset.hpp
    
```

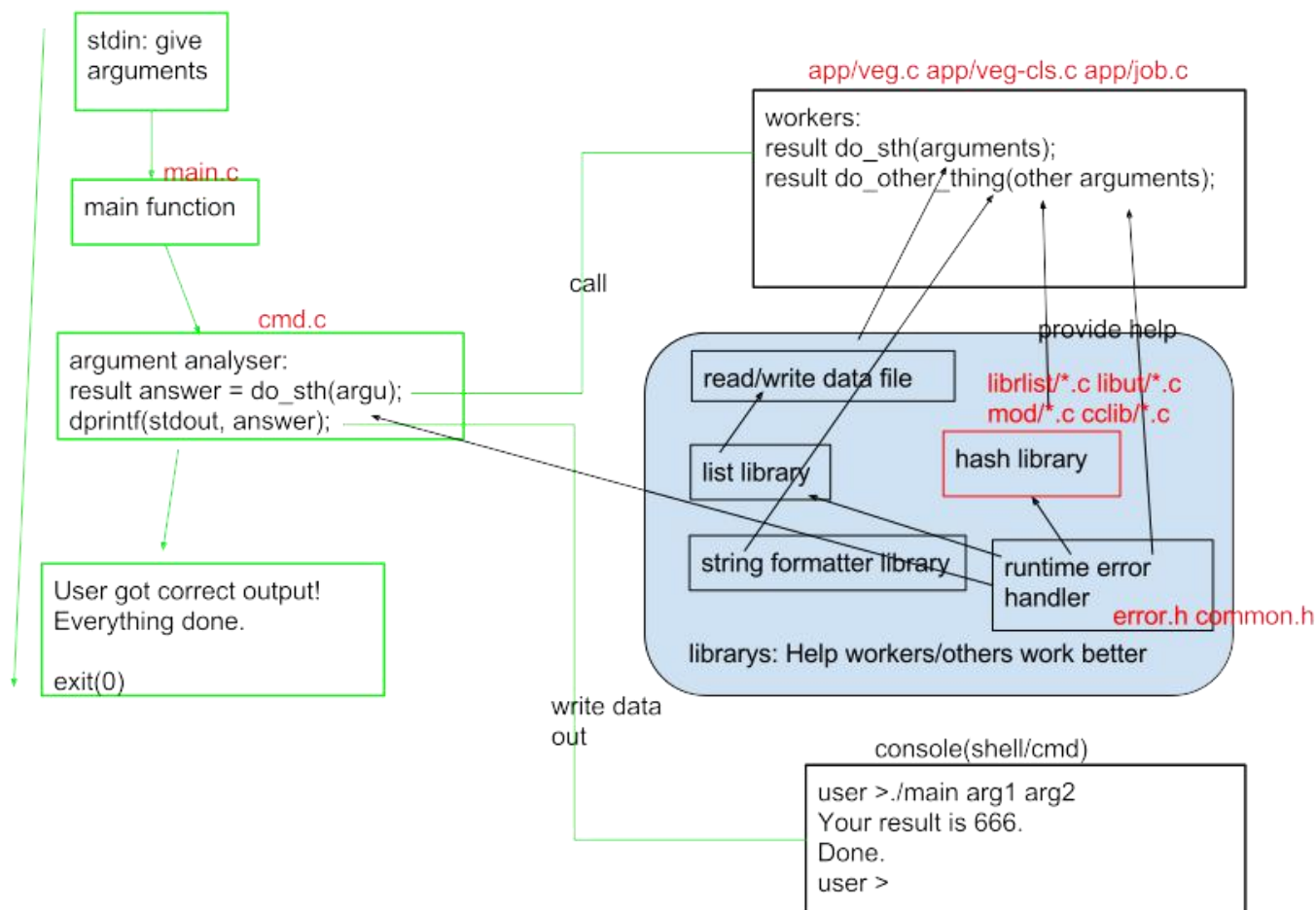
```

|   |— os.hpp
|   |— rwlock.hpp
|   |— sio.hpp
|— terminal.hpp
|— threadPool.hpp

```

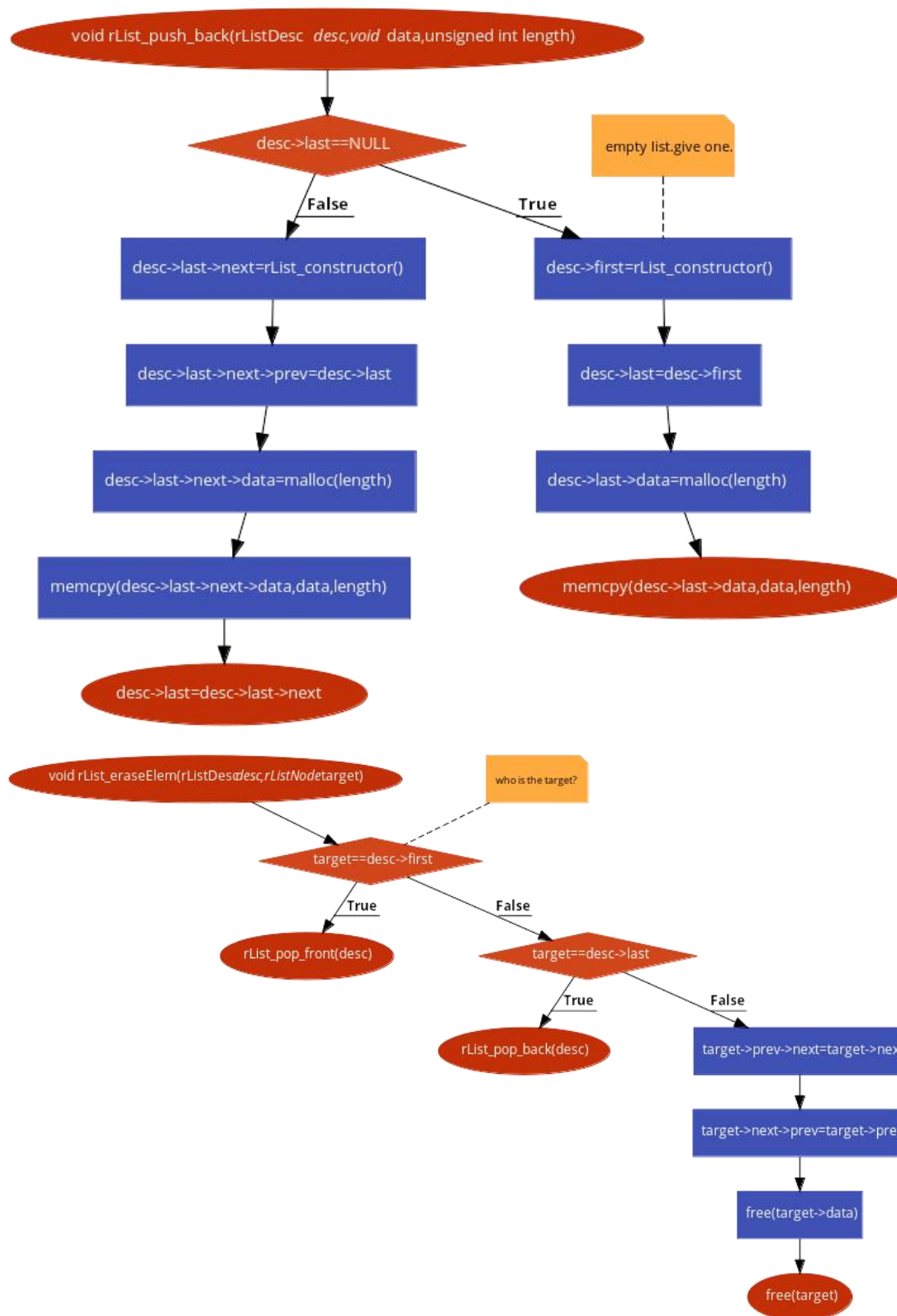
下面是程序运行的大致思路。绿色部分表示程序运行流程，黑色箭头表示依赖树(或依赖图)，红色文字表示对应的实现的大致源文件位置。

run this program from console



由于 filter 的存在，ls 操作将会稍有不同。

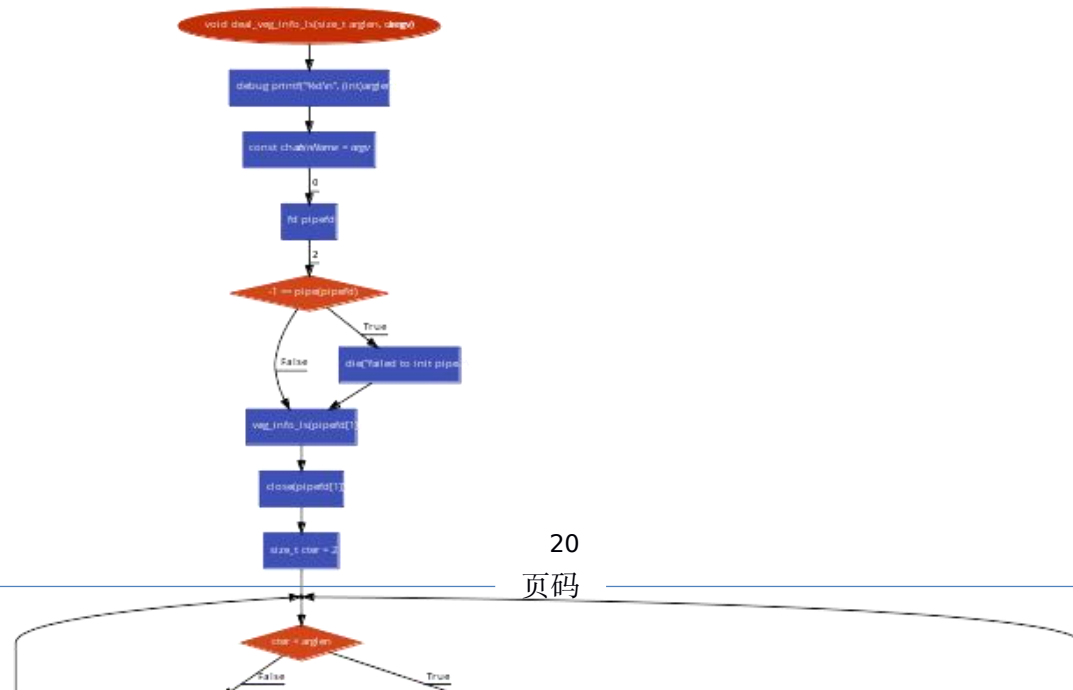
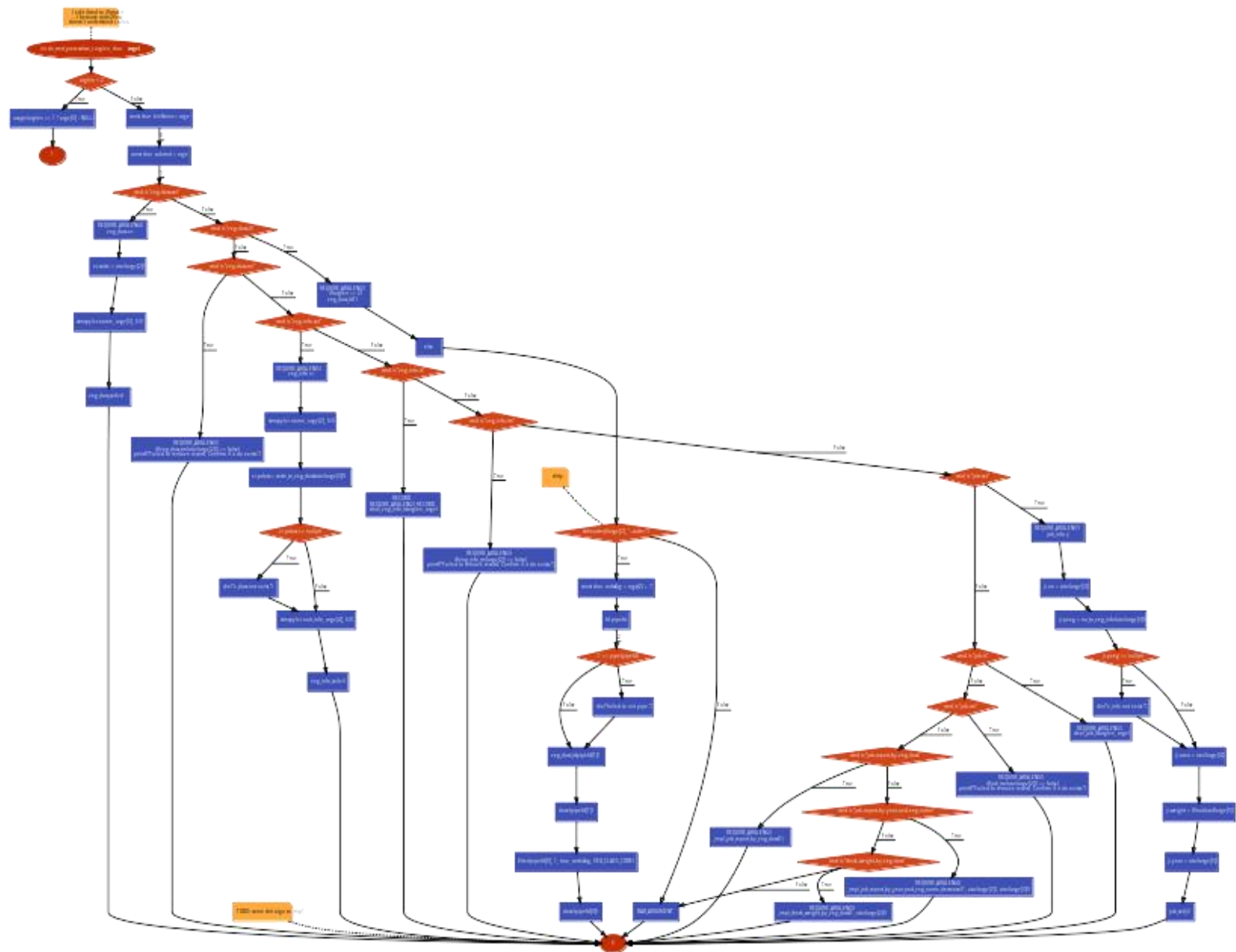
下面是被要求的 librlist 流程图。由于只有 rList_push_back, rList_for_each 和 rList_remove_item 被使用过，因此只列举这些方法的流程图。



可能值得特别说明的是 `do_cmd_process`。它进行语法解析和子操作分类，并进行管道对接和多次过滤等 `unix` 哲学常用操作。给出流程图。

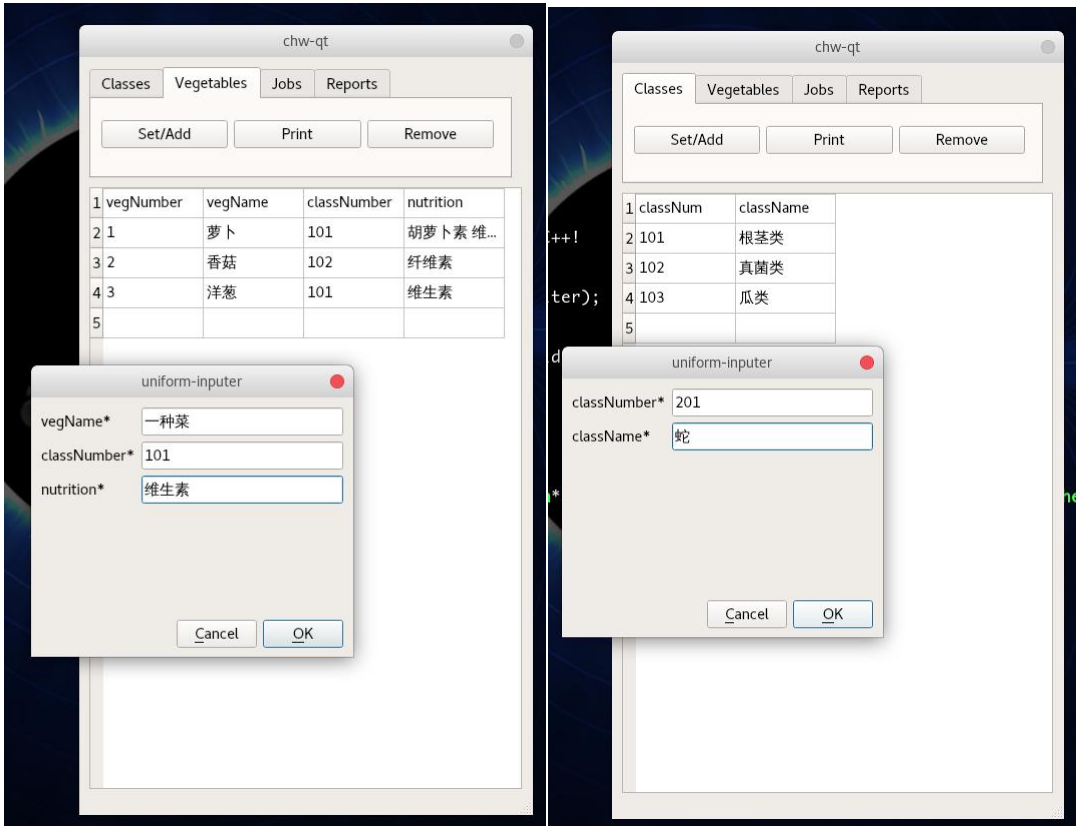
(可前往 <https://drive.recolic.net/d/93e93777d9>/获得无法打印清楚的原矢量图)

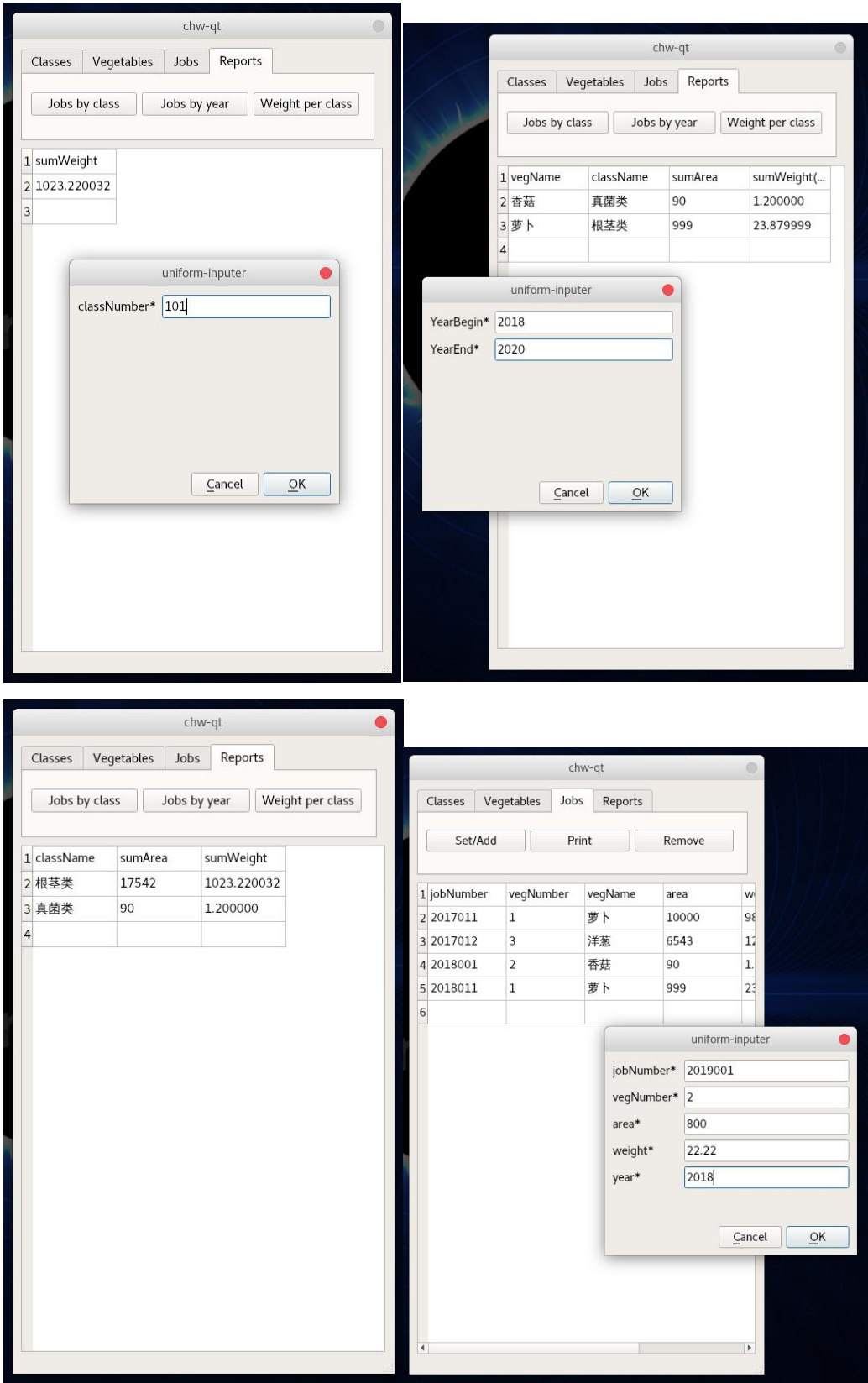
将上图中的 `deal_.*_ls` 分别予以展开



4 测试报告

首先，为了展示系统实现了所要求的所有功能，对前端进行演示。





其次，简单展示 web socket server(wserver), backend(chw-cli), web interface(client.min.html).

Select your operation *

Gen work report by vegetable class ▼

Submit

clear

output here:

200 Operation success.
1|萝卜|101|胡萝卜素 维生素A
2|香菇|102|纤维素 3|洋葱|101|维生素
200 Operation success.
根茎类|17542|1023.220032
真菌类|90|1.200000

Select your operation *

Set(add/edit) a vegetable info ▼

vegName *

金针菇

classNum *

103

neulInfoStr *

纤维素

Submit

clear

output here:

200 Operation success.

websocket server 与前端的交互:

```
root@archlinux:~/wserver (master)~# [2017-09-10 20:38:30] [connect] WebSocket Connection [::ffff:127.0.0.1]:33314 v13 Mozilla/5.0 (X11; Linux x86_64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/61.0.3163.79
CMD ARRIVED:veg-info-1s
[2017-09-10 20:38:38] [frame_header] Dispatching write containing 1 message(s) containing 2 header bytes and 83 payload bytes
[2017-09-10 20:38:38] [frame_header] Header Bytes:
[0] (2) 81 53
[2017-09-10 20:38:38] [frame_payload] Payload Bytes:
[0] (83) [1] 1|萝卜|101|胡萝卜素 维生素A
2|香菇|102|纤维素
3|洋葱|101|维生素
CMD ARRIVED:job-report-by-veg-class
[2017-09-10 20:38:44] [frame_header] Dispatching write containing 1 message(s) containing 2 header bytes and 50 payload bytes
[2017-09-10 20:38:44] [frame_header] Header Bytes:
[0] (2) 81 32
[2017-09-10 20:38:44] [frame_payload] Payload Bytes:
[0] (50) [1] 根茎类|17542|1023.220032
真菌类|90|1.200000
CMD ARRIVED:veg-info-set 金针菇 103 纤维素
[2017-09-10 20:39:27] [frame_header] Dispatching write containing 1 message(s) containing 2 header bytes and 0 payload bytes
[2017-09-10 20:39:27] [frame_header] Header Bytes:
[0] (2) 81 00
[2017-09-10 20:39:27] [frame_payload] Payload Bytes:
[0] (0) [1]
[2017-09-10 20:40:04] [control] Control frame received with opcode 8
[2017-09-10 20:40:04] [frame_header] Dispatching write containing 1 message(s) containing 2 header bytes and 2 payload bytes
[2017-09-10 20:40:04] [frame_header] Header Bytes:
[0] (2) 88 02
[2017-09-10 20:40:04] [frame_payload] Payload Bytes:
[0] (2) [8] 03 E9
[2017-09-10 20:40:04] [error] handle_read_frame error: websocketpp.transport:7 (End of File)
[2017-09-10 20:40:04] [disconnect] Disconnect close local:[1006,End of File] remote:[1001]
```

可以独立运行和测试，便于通过 shell 扩展的后端：

```
recolic@RECOLICPC ~/C/c/bin (master)> ./chw-cli
Usage:
./chw-cli <sub-command> <required arguments ...> [optional arguments ...]

sub-commands:
veg-class-set <classNum> <className>
veg-class-ls [--code=...]
veg-class-rm <classNum>
veg-info-set <vegName> <classNum> <neuInfoStr>
veg-info-ls [--name-keyword=...] [--neu-keyword=...] [--class-code=...]
veg-info-rm <vegName>
job-set <jobNum> <vegNum> <area> <weight> <year>
job-ls [--veg-name-keyword=...] [--veg-name=...] [--year=...]
job-rm <jobNum>

job-report-by-veg-class #print className+sumArea+sumWeight
job-report-by-year-and-veg-name <yearBegin> <yearEnd(NOT included)> #print vegName+className+sumArea+sumWeight(decrease)
check-weight-by-veg-class <classNum> #print sumWeight
Please use veg-info-ls to print veg info by neu keyword.

recolic@RECOLICPC ~/C/c/bin (master) [1]> ./chw-cli veg-info-ls
1|萝卜|101|胡萝卜素 维生素A
2|香菇|102|纤维素
3|洋葱|101|维生素
4|金针菇|103|纤维素
recolic@RECOLICPC ~/C/c/bin (master)> ./chw-cli veg-info-ls --neu-keyword='维生素'
1|萝卜|101|胡萝卜素 维生素A
3|洋葱|101|维生素
```

能够直接运行在后端的自动化的压力测试脚本保证了程序的可靠性。

```
>>> src/test/test.sh
#!/bin/bash

echo 'Here is test script for recolic/chw, by Recolic Keghart, Mozilla
licensed.'
[[ $1 == '' ]] && echo 'Usage: ./test.sh <bin dir> [disable stdout
redirect]' && exit 1

bindir=$1
[[ $2 != '' ]] && redir="/dev/null" || redir="/dev/fd/1"

function assert () {
    CMD="$*"
    $CMD
    RET_VAL=$?
    if [ $RET_VAL != 0 ]; then
        echo "Assertion failed: $CMD returns $RET_VAL."
        exit $RET_VAL
    fi
}

function randstr4 () {
    < /dev/urandom tr -dc _A-Z-a-z-0-9 | head -c${1:-4};echo;
}

function randstr32 () {
    < /dev/urandom tr -dc _A-Z-a-z-0-9 | head -c${1:-32};echo;
}

function randint () {
    echo "$(( ( RANDOM % 100 ) + 1 ))"
}

function test_vcset () {
    ./chw-cli veg-class-set $(randint) $(randstr32)
}

function test_viset () {
    ./chw-cli veg-info-set $(randstr32) $(randint) $(randstr32)
}

function test_jset () {
    ./chw-cli job-set $(randint) $(randint) $(randint)
```

```

"$(randint).$(randint)" $(randint)
}

function test_vcrm () {
    ./chw-cli veg-class-rm $(randint)
}

function test_virm () {
    ./chw-cli veg-info-rm $(randint)
}

function test_jrm () {
    ./chw-cli job-rm $(randint)
}

function test_ls () {
    echo 'performing ls...'
    echo 'veg-cls->'
    ./chw-cli veg-class-ls
    echo 'veg-inf->'
    ./chw-cli veg-info-ls
    echo 'job->'
    ./chw-cli job-ls
}

assert cd $bindir
assert ./clear_database.sh

echo 'perform 1000 veg class set...'
for i in {1..1000}
do
    echo -ne "\r$i"
    assert test_vcset > $redir
done
echo ' '
assert test_ls

echo 'perform 30 veg class rm, may fail...'
for i in {1..30}
do
    echo -ne "\r$i"
    test_vcrm
done
echo ' '
assert test_ls

echo 'perform 1000 veg class set...'
for i in {1..1000}
do
    echo -ne "\r$i"
    assert test_vcset > $redir
done
echo ' '

echo 'perform 300 veg info set...'
for i in {1..300}
do
    echo -ne "\r$i"
    test_viset > $redir
done
echo ' '
assert test_ls

echo 'Because veg name must be given, skipping veg-info-rm-test'

echo 'perform 1000 job set...'
for i in {1..1000}
do
    echo -ne "\r$i"
    test_jset > $redir
done
echo ' '
assert test_ls

echo 'perform 50 job rm, may fail...'
for i in {1..50}
do

```

```

    echo -ne "\r$i"
    test_jrm
done
echo ' '
assert test_ls

echo 'perform 1000 job set...'
for i in {1..1000}
do
    echo -ne "\r$i"
    test_jset > $redir
done
echo ' '

echo 'test done. Try filtered ls/summery manually.'
cd -
<<< test.sh

```

多次随机化的压力测试完全通过。共计 180000 次随机插入删除测试完全通过。

这表明了算法的可靠性。

	jobNumber	vegNumber	vegName	area
1				
2	62	51	UWb14Gt9...	72
3	38	91	X3vwDP5SF...	21
4	76	55	-rvHqwO0d...	100
5	19	23	4gcdADMY1...	1
6	37	7	ODWHiliOq...	19
7	3	23	4gcdADMY1...	38
8	59	33	7jmWHdGxf...	20
9	46	57	DiE3gw53m...	34
10	12	85	p91sDVLYN...	86
11	75	45	E852ffLunZ...	61
12	77	78	dtkX57u6h...	92
13	15	13	mvLkdRFL8...	94
14	20	75	jkBwaFG7n...	91
15	54	60	WROnbSJ...	15
16	48	43	s0eyNZDIY...	89
17	87	48	fh4D51q8Kj...	29
18	10	40	X98Z0whO...	100
19	73	21	13yZx87Pak...	38
20	67	2	B6FvCQ7mf...	1

travis-ci 的实时在线编译测试结果可以反映运行环境的复现较稳定。

5 项目管理

本程序采用功能点技术估算软件规模及工作量，根据系统的设计：输入的项数为 26，其中简单级为 5 项，平均级为 10 项，复杂级为 11 项；输出项数为

53, 其中简单级为 10, 平均级为 30, 复杂级为 13 项; 查询项数为 49 项, 其中简单级为 13 项, 平均级为 21 项, 复杂级为 15 项; 主文件项数为 4 项, 其中简单级为 2 项, 平均级为 2 项, 复杂级为 0 项; 外部接口数为 4 项简单级为 4 项, 平均级为 0 项, 复杂级为 0 项。根据功能点 UFP 计算公式, 计算出本程序的 $UFP=669$ 。

根据对系统技术因素的分析, 数据通信 $F1=2$, 分布式数据处理 $F2=0$, 性能标准 $F3=3$, 高负荷的硬件 $F4=0$, 高处理率 $F5=2$, 联机数据输入 $F6=1$, 终端用户效率 $F7=4$, 联机更新 $F8=0$, 复杂的计算 $F9=1$, 可重用性 $F10=0$, 安装方便 $F11=3$, 操作方便 $F12=3$, 可移植性 $F13=5$, 可维护性 $F14=3$ 。根据技术复杂因子 TCF 的计算公式估算系统的 TCF 为 0.92。

所以系统的功能点数 $FP=669*0.92=615.48$ 。

对于系统工作量的估算, 运用静态单变量模型中的 Kemerer 模型计算, 综合上述 FP 数据,

根据获得了的项目历史生产率数据, 其中平均生产率为 50FP/PM, 平均生产工资为 30 元/FP, 平均工资成本为 1000 元/人。

所以工作量的估算为 $E=12.30PM$, 3 人完成, 工期为 4.5 个月, 总成本的估算为 21464.4 元

项目人员分配为民主分权式。软件资源小粒度复用, 开发环境为 gcc8.2, clang7.0, archlinux, 联机环境。硬件资源为计算机 2 台。

项目任务有: 算法设计 1.5 个月, 原型开发以模块开发为主系统分为 4 个模块, 系统测试 1.5 个月, 文档写作 0.5 个月等。

6 体会与建议

`cmake` 和 `travis-ci` 都曾被我用在生产环境的 `project` 中,并表现地令人满意。`project` 的设计遵从 Mike Gancarz 归纳的 `unix` 一贯的设计理念,模块之间要求低耦合,前后端严格分离,保证了 `project` 的 `portable` 和 `extensible`,极大的方便了自动测试和生产环境部署(尽管课设并没有为生产环境做过多的考虑,但前后端分离是最基本的要求)。在开发速度和代码质量之间作出了恰当的权衡,基本遵守 `Google C++` 代码风格规范。除了文档要求必须自己实现的数据结构外,最大限度复用已有代码以增加开发速度和程序可靠性与鲁棒性,因此我打包了所需的大多数依赖(除 `boost_system`,被静态链接)并用 `cmake` 依次自动编译,这也是 `project` 中代码总量 200k 行的原因。其中定义业务逻辑的代码仅 707 行。

发现并最小复现了 `libccl` 的 `hash` 实现的一个 `bug`,学习了 `libut` 的纯宏的 `template` 实现,然后看了一下 `glibc` 的 `list` 中的 `template` 实现,尝试了 `C` 调用其他编译型语言的 `dylib` 的 `trick`(`CPython`, `CGo`)和 `BashInC` 的 `trick`(最终并没有用到),这也是 `project` 中涉及了多种语言代码的原因。

Small is beautiful.

Make each program do one thing well.

Build a prototype as soon as possible.

Choose portability over efficiency.

Store data in flat text files.

Use software leverage to your advantage.

Use shell scripts to increase leverage and portability.

Avoid captive user interfaces.

Make every program a filter.

参考文献

- [0]https://en.wikipedia.org/wiki/Unix_philosophy
- [1]Google C++ Style Guide:<https://google.github.io/styleguide/cppguide.html>
- [2]libccl:URL:<https://github.com/Lupus/ccl>
- [3]libccl:URL:<http://www.open-std.org/jtc1/sc22/wg14/www/docs/n1625.pdf>
- [4]libccl:URL:<http://www.cs.virginia.edu/~lcc-win32/ccl/ccl.html>
- [5]libut:URL:<https://troydhanson.github.io/uthash/>
- [6]URL:<http://en.cppreference.com/w/>
- [7]URL:<http://www.cplusplus.com/reference/>
- [8]ISO/IEC 9899:201x:<http://www.open-std.org/jtc1/sc22/wg14/www/docs/n1548.pdf>
- [9]ISO/IEC 14882:2011:<http://www.open-std.org/jtc1/sc22/wg21/docs/papers/2012/n3337.pdf>
- [10]ISO/IEC 14882:2014:<https://www.iso.org/obp/ui/#iso:std:iso-iec:14882:ed-4:v1:en>
- [11]ISO/IEC 14882:2017:<http://www.open-std.org/jtc1/sc22/wg21/docs/papers/2017/n4659.pdf>
- [12]IEEE Std 1003.1-2008, 2016
Edition:<http://pubs.opengroup.org/onlinepubs/9699919799/functions/contents.html>
- [13]Linux Manual (man 2, man 3)
- [14]URL:<https://linux.die.net/>
- [15]URL:<http://man7.org/linux/man-pages/>
- [16]URL:<https://www.qt.io/>
- [17]URL:<https://en.wikipedia.org/wiki/WebSocket>
- [18]URL:<https://tools.ietf.org/html/rfc6455>
- [19]URL:<https://github.com/zaphoyd/websocketpp>
- [20]URL:<https://www.gnu.org/software/bash/manual/bash.html>
- [21]URL:https://wiki.gentoo.org/wiki/Main_Page
- [22]URL:https://wiki.archlinux.org/index.php/Main_page
- [23]URL:<http://libcello.org>