极客班第一期 C++专业期末综合测试题

[TOC]

说明:

1. 极客班第一期学员可通过项目实践(样本项目或者自选项目)或者此试题来参加期 末考评。
2. 答案提交截止日期为2015年10月18日 23:30。
3. 请将答案提交到自己在极客班github的C++ Homework资源库里。 地址:https://github.com/GeekBand/GeekBand-CPP-1501-Homework 请在以自己学号命名的文件夹中,创建文件夹“C++综合测试”,并将作业文档 (PDF格式)和代码通过pull request的方式提交。 请在作业文档中注明题目和对应的代码文件夹(部分题目所需的关键代码可以 写在作业文档里),方便老师批改。
4. 此试题分数(或项目实践)占整个专业考评的70%;课程测验(平时作业),占 20%;课堂优秀表现(分享、讨论、创新),占10% 。

解答

以下解答中出现的代码路径均相对于 `*G2015010320/C++综合测试/src`

PDF 文档：解答.pdf

面向对象与设计

题目1:

设计一个多边形 Polygon,其由若干个点连成线。为其实现构造函数,拷贝构 造函数,复制操作符,析构函数。

```
class Shape{
  int no; //形状编号
};

class Point{ int x;
  int y;
};
class Polygon: public Shape{
  Point* points;
};
```

解答:

- 实现代码 `OOD/Shape.h` `OOD/Polygon.h`
- 演示代码 `OOD/1.cpp`

```
Polygon pg1(points);
std::cout << pg1 << std::endl;
Polygon pg2(pg1);
// update the 2nd point of pg2
pg2.at(1) = Point(100, 100);
std::cout << pg2 << std::endl;
```

- 演示运行和输出:

```
$ g++ -Wall 1.cpp -o 1 ; time ./1
[Polygon 0 : (P: 0, 2), (P: 0, 1), (P: 0, 0), (P: 1, 2), (P: 1, 1), (P: 1,
0), (P: 2, 2), (P: 2, 1), (P: 2, 0) ]
[Polygon 1 : (P: 0, 2), (P: 100, 100), (P: 0, 0), (P: 1, 2), (P: 1, 1), (P:
1, 0), (P: 2, 2), (P: 2, 1), (P: 2, 0) ]

real    0m0.004s
user    0m0.001s
sys 0m0.002s
```

- `Polygon` 继承自 `Shape` 基类，实现其构造、拷贝构造、赋值操作符、析构函数如下:

```
// note: it hands over the life of points
class Polygon : public Shape
{
private:
  std::vector<const Point *> ps_;

public:
  Polygon(const std::vector<const Point *> &ps) : ps_(ps) { }
  Polygon(const Polygon &p) : ps_(__deep_copy(p.points())) { }
  Polygon & operator=(const Polygon &p) {
    if (this == &p) {
      return *this;
    }

    __clean(ps_);
    ps_ = __deep_copy(p.points());
    return *this;
  }

  virtual ~Polygon() {
    __clean(ps_);
  }

public:
```

```
  const std::vector<const Point *> points() const {
    return ps_;
  }

  // the only data update interface for changing the
  // Point at specified position
  // usage: Polygon.at(i) = Point(x,y);
  Point & at(int idx) const {
    assert(idx > 0 && idx < ps_.size());
    return const_cast<Point &>(*ps_.at(idx));
  }

private: // some helper methods
  void __clean(std::vector<const Point *> &ps) {
    for (int i = 0; i < ps.size(); ++i) {
      const Point *p = ps.at(i);
      delete p;
    }
    ps.clear();
  }

  std::vector<const Point *> __deep_copy(const std::vector<const Point *>
&ps) {
    std::vector<const Point *> copy;
    for (int i = 0; i < ps.size(); ++i) {
      const Point *p = ps.at(i);
      copy.push_back(new Point(*p));
    }
    return copy;
  }
};

inline std::ostream &
operator<< (std::ostream &os, const Polygon &pg) {
  os << "[Polygon " << pg.id() << " : ";
  char dm[3] = { '\0', ' ', '\0' };
  for (int i = 0; i < pg.points().size(); ++i) {
    const Point *p = pg.points().at(i);
    os << dm << *p;
    dm[0] = ',';
  }
  os << " ]";
  return os;
}
```

题目2:

为Point类设计一个数据绑定机制,当其坐标x或y被更改时,可以通知外界 其更改的过程(即从旧值改为新值)。

将更改过程打印在控制台上。 考虑使用松 耦合设计(即未来有可能将更改过程显示在任何界面上)。

```
class Point {
  int x:
  int y;
}
```

解答:

- 实现代码路径 `OOD/Point.h` `OOD/Observable.h` `OOD/Handles.h` `OOD/Handles.h`
- 演示代码 `OOD/2.cpp`

```
TextBox box; // a TextBox somewhere
Point origin(1, 2);

// define a observable model(point) with origin value(1, 2)
Observable<Point> op(origin);
// define and bind an observer handler to above observable model(point)
std::auto_ptr< Handler<Point> > console_h(Handlers::of<Point>(std::cout));
std::auto_ptr< Handler<Point> > box_h(Handlers::of<Point>(box));
Binder::bind(*console_h.get(), op);
Binder::bind(*box_h.get(), op);

// print the origin value(1, 2)
std::cout << op.getValue() << std::endl;

Point newValue(1, 3);
// set a new value(1, 3) to emit changed-event
op.setValue(newValue);

// print the new value(1, 3)
std::cout << op.getValue() << std::endl;
```

- 演示运行和输出:

```
$ g++ -Wall 2.cpp -o 2 ; time ./2
[Observable]    bind    OStream
[Observable]    bind    TextBox
(P: 1, 2)
[OStream]   value-changed to    (P: 1, 3)
[TextBox]   value-changed to    (P: 1, 3)
(P: 1, 3)
[Handler]   TextBox dtor
[Handler]   OStream dtor
[Observable]    dtor

real    0m0.004s
user    0m0.001s
sys 0m0.002s
```

STL与泛型编程

题目1：

给定一个 vector:v1 = [0, 0, 30, 20, 0, 0, 0, 0, 10, 0],希望通过 not_equal_to 算法找到到不为零的元素,并复制到另一个 vector: v2

解答:

- 实现代码 `STL/1.cpp`：

```
// copy values of not equal to 'target' to the dest vector
template<typename T>
void not_equal_to(const std::vector<T> &src, std::vector<T> &dest, const T
&target) {
  for (typename std::vector<T>::const_iterator it = std::begin(src);
    it != std::end(src); ++it) {
    if (*it != target) {
      dest.push_back(*it);
      }
  }
}

template<typename T>
inline std::ostream &
operator<< (std::ostream &os, const std::vector<T> &v) {
  os << "[ ";
  char dm[3] = { '\0', ' ', '\0' };
  for (typename std::vector<T>::const_iterator it = std::begin(v);
    it != std::end(v); ++it) {
    std::cout << dm << *it;
    dm[0] = ',';
  }
  return os << " ]";
}
```

- 演示代码:

```
const int N = 10;
int arr[] = { 0, 0, 30, 20, 0, 0, 0, 0, 10, 0 };
std::vector<int> v1(arr, arr + N);

std::cout << v1 << std::endl;
std::vector<int> v2;
not_equal_to(v1, v2, 0);
std::cout << v2 << std::endl;
```

- 演示运行和输出:

```
$ g++ -Wall 1.cpp -o 1; time ./1
[ 0, 0, 30, 20, 0, 0, 0, 0, 10, 0 ]
[ 30, 20, 10 ]

real    0m0.003s
user    0m0.001s
sys 0m0.002s
```

题目1:

为以下 Programmer 对象提供一个基于 Id 并且升序的仿函数 ProgrammerIdGreater,使得 Programmer 对象可以在 set 中以 Id 排序存放

解答:

- 实现代码 `STL/2.cpp`:

```
class Programmer
{
private:
  const int id_;
  std::string name_;

public:
  explicit Programmer(const int &id, const std::string &name)
    : id_(id), name_(name) { }

  int id() const { return id_; }
  std::string name() const { return name_; }
  void setName(std::string name) {
    name_ = name;
  }

public:
  virtual ~Programmer() { }
};

// provides the sorted and unique
// capacity for container of `Programmer`
// with id in desc order
class ProgrammerIdGreater
{
public:
  bool operator()(const Programmer *p1, const Programmer *p2) const {
    assert(p1 != NULL && p2 != NULL);
    return (p1->id() - p2->id()) > 0;
  }
};

// provides the sorted and unique
// capacity for container of `Programmer`
// with name in asc order
class ProgrammerNameLess
{
public:
  bool operator()(const Programmer *p1, const Programmer *p2) const {
    assert(p1 != NULL && p2 != NULL);
    return std::strcmp(p1->name().c_str(), p2->name().c_str()) < 0;
```

```
  }
};

class PrintForEach
{
public:
  void operator()(const Programmer *p) {
    assert(p != NULL);
    std::cout << p << ", ";
  }
};

class deleteForEach
{
public:
  void operator()(const Programmer *p) {
    assert(p != NULL);
    delete p;
  }
};

// helper of printing Programmer
inline std::ostream &
operator<< (std::ostream &os, const Programmer *p) {
  return os << "[" << p->id() << "]: " << p->name();
}

// helper of printing containers.
template<class Ch, class Tr, class Co>
std::basic_ostream<Ch, Tr>& operator<<(std::basic_ostream<Ch, Tr>& stream,
Co& c) {
  stream << "{ ";
  std::copy(c.begin(), c.end(), std::ostream_iterator<typename Co::key_type>
(stream, ", "));
  return stream << "}";
}

inline void
initArr(const Programmer *ps[]) {
  ps[0] = new Programmer(1, "Scott Meyers");
  ps[1] = new Programmer(2, "Martin Fowler");
  ps[2] = new Programmer(3, "Bill Gates");
  ps[3] = new Programmer(4, "P.J. Plaught");
  ps[4] = new Programmer(4, "Stanley B. Lippman"); // emulate id conflicts
  ps[5] = new Programmer(6, "Andrei Alexandrescu");
}

// It's a demostraction, you should not use this to delete anything
inline void
cleanSet(const std::set<const Programmer *, ProgrammerIdGreater>& c) {
  std::for_each(c.begin(), c.end(), deleteForEach());
```

```
}

inline void
cleanArr(const Programmer *ps[], int size) {
  const int N = size;
  for (int i = 0; i < N; ++i) {
    if (ps[i] != NULL) delete ps[i];
  }
}
```

- 演示代码:

```
const int N = 6;
const Programmer *ps[N] = { NULL };
initArr(ps);

std::set<const Programmer *, ProgrammerIdGreater> p_set_A(ps, ps + N);
std::cout << "Origin set A: " << p_set_A << std::endl;
// find the Programmer with id = 3
{
std::auto_ptr<Programmer> target(new Programmer(3, "David Vandevoorde"));
std::set<const Programmer *, ProgrammerIdGreater>::iterator found =
    p_set_A.find(target.get());
if (found != std::end(p_set_A)) { // found it and modify then
    // this const cast is necessary, since we need to change the name here
    const_cast<Programmer *>(*found)->setName(target->name());
}
}
std::cout << "Modified set A: " << p_set_A << std::endl;

std::set<const Programmer *, ProgrammerNameLess> p_set_B(ps, ps + N);
std::cout << "Modified set B: " << p_set_B << std::endl;

cleanArr(ps, N);
```

- 演示运行和输出:

```
$ g++ -Wall 2.cpp -o 2; time ./2
Origin set A: { [6]: Andrei Alexandrescu, [4]: P.J. Plaught, [3]: Bill
Gates, [2]: Martin Fowler, [1]: Scott Meyers, }
Modified set A: { [6]: Andrei Alexandrescu, [4]: P.J. Plaught, [3]: David
Vandevoorde, [2]: Martin Fowler, [1]: Scott Meyers, }
Modified set B: { [6]: Andrei Alexandrescu, [3]: David Vandevoorde, [2]:
Martin Fowler, [4]: P.J. Plaught, [1]: Scott Meyers, [4]: Stanley B.
Lippman, }

real    0m0.003s
user    0m0.001s
sys 0m0.002s
```

算法与系统设计

题目1:

Moving Average

解答:

- 实现代码 `ALG/ma.cpp`:

```
class MovingAverage
{
private:
  double sum_;
  const int period_;
  std::deque<double> window_;

public:
  explicit MovingAverage(int period) : period_(period) { }
  double next(double num) {
    sum_ += num;
    window_.push_back(num);
    if (window_.size() > period_) {
      sum_ -= window_.front();
      window_.pop_front();
    }

    return sum_ / window_.size();
  }

public:
  virtual ~MovingAverage() { }
private:
  MovingAverage(const MovingAverage &);
  MovingAverage & operator= (const MovingAverage &);
};
```

- 演示代码:

```
MovingAverage ma(2);
std::cout << ma.next(1) << ", " << ma.next(3) << ", " << ma.next(4) <<
std::endl;
```

- 演示运行和输出:

```
$ g++ -Wall ma.cpp -o ma ; time ./ma
1, 2, 3.5

real    0m0.003s
user    0m0.001s
sys 0m0.001s
```

题目2:

Total Difference Strings

解答:

- 实现代码 `ALG/diffstr.cpp`:

```
class __str_compare_q2 {
public:
  bool operator() (const std::string &s1, const std::string &s2) {
    const char *c1 = s1.c_str();
    const char *c2 = s2.c_str();

    if (s1.length() != s2.length()) {
      return true; // special case, just for q2's compare operator
    }

    const int LEN = s1.length();
    for (int i = 0, j = LEN - 1; i < LEN && j >= 0; ++i, --j) {
      if (c1[i] != c2[i] && c1[i] != c2[j]) {
        return true;
      }
    }
    return false;
  }
};

class TDS // TotalDiffString
{
public:
  static int count(const std::string str_arr[], int size) {
    std::set<std::string, __str_compare_q2> ss(str_arr, str_arr + size);
    return ss.size();
  }

private:
  TDS();
};
```

- 演示代码:

```
const int N = 4;
const std::string str_arr[] = { "abc", "cba", "abc", "Aaa" };
int size = TDS::count(str_arr, N);
std::cout << size << std::endl;
```

- 演示运行和输出:

```
g++ -Wall diffstr.cpp -o diff ; time ./diff
2

real    0m0.003s
user    0m0.001s
sys 0m0.002s
```

题目3:

Binary Tree Print

解答:

- 实现代码 `ALG/bt.cpp`:

```
void __print_path(const std::vector<Node *> &path) {
  std::cout << path;
}

void __print_routs(Node *root, std::vector<Node *> &path) {
  if (root != NULL) {
    path.push_back(root);
    if (root->left_ == NULL && root->right_ == NULL) {
      __print_path(path);
      std::cout << "\n----------" << std::endl;
    } else {
      __print_routs(root->left_, path);
      __print_routs(root->right_, path);
    }
    path.pop_back();
  }
}

void printBtree(const BTree &tree) {
  std::vector<Node *> path;
  __print_routs(tree.root, path);
}
```

- 演示代码:

```
BTree tree;
init(tree);
printBtree(tree);
```

- 演示运行和输出:

```
$ g++ -Wall bt.cpp -o bt ; time ./bt
{ 3, 1, 2, }
----------
{ 3, 5, 6, 8, }
----------
{ 3, 5, 7, }
----------

real    0m0.003s
user    0m0.001s
sys 0m0.002s
```

题目4:

如何设计一个购物车,可以从商品列表中添加商品,修改数量,生成订单,如果商 品数量在1亿以上,如何设计架构保证安全稳定的, (multi-tier, MVC, SOA)

bonus:如何设计抢购页面, suppose we have limited products and time

解答:

实现 `ALG/Design Shopping Cart.md`。
问题分析:

1. 明确`shopping cart`是针对某个用户唯一，所以数据的应该可以通过用户UID索引查询。
2. 明确每个`shopping cart`应该具备一个商品列表，而且支持“商品列表中添加商品, 修改数量”。
3. 明确用户需要从`shopping cart`生成订单，所以购物车服务应该支持快速生成订单的功能。
4. 对于抢购类商品提供“抢购”功能，需要在购物车服务之上扩展出一个支持异步消息队列的中间服务。

系统可能的层次结构和实施方案:

1. WEB接入层（multi-tier & MVC）：提供购物车浏览操作的WEB服务WEB1，抢购页面WEB2。
2. SS1: 购物车中间业务服务, 考虑支持通过用户UID的进行负载均衡（multi-tier & SOA）。
3. SS2: 抢购功能中间业务服务, 考虑支持通过用户UID的进行负载均衡（multi-tier & SOA）（可以考虑基于异步分布式消息队列）。
4. CH：用户购物车分布式Cache数据视图，Key是用户UID，Value是购物车商品列表。
5. PL：持久化层：提供用户购物车的持 久化视图，考虑使用mysql，考虑基于用户UID进行散表散库，针对用户id构建索引，用于减轻查询负载。
6. 在中间业务服务 SS1 中，对于购物车的修改、添加以及生产订单操作需要加上分布式锁（用户UID为粒度），用于防止用户多终端并行操作导致出现不一致的状态。

7. SS2 提供“抢购”排队“请求缓冲”的功能，其一旦“抢购”成功，调研 SS1 的接口将商品加入购物车。
8. WEB层提供无状态的查询和更新接口，考虑通过反向代理（nginx）对WEB请求提供负载均衡。
9. SS1 同步操作 CH，异步操作PL，最终保持CH和PL数据视图的一致。
10. SS1-2，考虑利用分布式协调系统（zookeeper集群）进行服务寻址和服务监控，保证没有单点故障和高可用性。
11. 考虑通过SS1-2 的客户端（WEB）使用UID和最终得到服务地址列表进行调用的负载均衡。