SORAY CENGİZ
161024077

HW01:

5.1.5 boost-bimap-examples-cpp

5.1.5.1 initial_boost_bimap_example.cc

Bu kodda bidirectional map yapısı kullanılmıştır. Bidirectional map, unordered veya map yapısından farklı olarak key, value çiftinde key'in value veya value'nin key olabilmesini sağlamaktadır. Unordered veya map yapısında key kullanılarak value değiştirilebiliyor fakat value kullanılarak key değiştirilemiyordu bunu görebilmek için 6.sayfada 5.1.5.1 için Ekler başlığı altında EK 1 yazılıp denenmiştir.

Boost kütüphanesi içerisindeki bimap özelliğini kullanarak bir map içerisinde key, value çifti birbiri yerine kullanılabilecektir. Key value olarak veya value key olarak kullanılabilecektir.

In C++:

```
1  #include<iostream>
2  #include<cstring>
3  #include<string>
4  #include<iomanip>
5  #include<memory>
6  #include<sstream>
7  #include<map>
8  #include<boost/bimap.hpp>
9
10 namespace os_test {
11
12   class Container
13   {
14   public:
15     Container(int number = 0) : number_(number){}
16
17     ~Container() = default;
18
19     int get_number() { return number_; }
20
21   private:
22     int number_;
23
24   };// class
25
26   namespace common_typedefs {
27     using bimap_type
28       = boost::bimap<std::string, std::shared_ptr<os_test::Container>> ;
29
30     using value_type
31       = bimap_type::value_type;
32
33   } // namespace
34
```

Kodda amaçlanan şey map içerisine smart pointer tutturabilmek olduğu için bir class yapısı kullanarak bu class'ın pointer objelerini map'e value olarak (tam olarak value değil bu key de olabilecek) tutturacağız.

Bir namespace açılıyor (C++ OOP language olduğu için encapsulation prensibine uymak gerek) yapılacak işler bu namespace arasında yapılsın fonksiyonlar, class'lar bunun içinde yapılsın istenmektedir. Basit bir Container class'ı oluşturuldu bir int private değer tutuyor class içerisinde;

(NOT : Buradan sonra bahsedilecek satır numaraları bu dosya içerisindeki görsellerdeki bu kodda bulunan satır sayılarıdır bir sonraki ödevlerde okuma kolaylığı için PDF'deki satır numaraları kullanılacaktır.)

15.satırda bir default constructor ve constructor tek satırda yapılmış olarak gözükmektedir. (int number = 0) yazarak oluşturulacak obje bir ilk değer verilmeden oluşturulursa otomatik number = 0 olacaktır. Örneğin Container c_obj; bize c_obj'nin number'ının 0 olduğunu söyleyecektir.

17.satırda destructor 19.satırda ise getter yazılmıştır. Getter şart main içerisinde gidip de private member'ı çağıramayız. Private olan bir şeye doğrudan erişim yok getter, setter lazım.

Yeni bir namespace açılmış common_typedefs içerisinde tip tanımlamaları yapılıyor; namespace içerisinde bir namespace açılmış (nested namespace)

27.satırda using keyword'ü aracılığı ile bir alias yapılmak istenmiştir, artık kodda bimap_type görülen yerde sağda kalan uzun yapı algılanacaktır. boost namespace'i içerisinde bimap'classı yer almaktadır template parametresi olarak string ve container tipinde bir smart pointer alacaktır. boost::bimap bir class template'dır, bu class template'a parametreler sayesinde class template instance oluyor ve bimap_type bir tip olarak anlam kazanabiliyor.

30.satırda aynı şekilde uzun bir yapı yerine daha kısasını kullanmak istiyoruz ve boost::bimap tanımlanırken içerisinde value_type isminde bir tip tanımlanmıştır. 120.satırda value_type kullanılmıştır kullanım amacı map içerisine kaydedilecek ikilileri value_type(key, value) şeklinde kaydetmemiz beklenmektedir, yaptığı iş aslında gerçekten map içerisinde string ve smart pointer ikilisi mi kaydedilmeye çalışılıyor onu test etmektir gidip de value_type(int, smart pointer) olarak map içerisine kaydetmeye çalışsak bu satır hata verecektir.

```
35  namespace utilities {
36    using namespace os_test::common_typedefs;
37
38    void print_map
39    (
40      const bimap_type & current_map,
41      std::string        choice = "left"
42    )
43    {
44      using namespace std;
45      enum choices
46      {
47        left = 0,
48        right
49      };
50
51      std::map<std::string, int>
52        choices_map =
53          {
54            {"left" ,  choices::left},
55            {"right",  choices::right}
56          };
57
58      if( choices_map.find(choice) == choices_map.end() )
59        return;
60
61      int choice_current = choices_map[choice];
62
```

35.satırda bir nested namespace daha görüyoruz bu namespace içerisinde kullanılacak fonksiyonlar yazılacaktır.

36.satırda bir common_typedefs namespace'inde tanımladığımız typedef'leri (::) binary scope resolution operatörünü kullanarak bu namespace içerisine de çağırıyoruz. os_test::common_typedefs burada :: operatörünün görevi soldaki yere gidip yani os_test içerisinde common_typedefs'i bul ve süslü parantez içerisinde kalan kısmı utilities namespace'ine de tanıtmaktır.

38.satırda bir print_map fonksiyonu yazılmıştır giriş parametresi olarak bimap_type ve bir choice almaktadır burada choice'a default argüman olarak "left" atanmıştır yani bu fonksiyon main içerisinde sadece bimap_type tipinde bir argüman verilince de çalışacaktır. bimap_type tipindeki değişkeni bu fonksiyon içerisinde ne yaparsak yapalım değiştiremeyiz const olarak alınıyor ve aynı zamanda kopyalama olmaması için call by reference yapılıyor fonksiyon içerisinde bimap_type tipindeki değişkenin adresi kullanılarak işler yapılacaktır. İkinci parametre choice'a bağlı olarak yapılacak iş değişecektir;

44.satır olmasaydı cout ve endl'yi kullanamazdık.

45.satırda bir enumeration yapılmıştır sebebi daha sonra kullanılacak olan switch yapısına statement flow olarak string koyamamaktır. Bu yapıyı kullanarak left ve right stringlerini bir integer değerler gibi bakabiliyoruz. 50.satıra cout << choices::left << choices::right yazsaydım sırası ile 0 ve 1 ekrana bastıracaktı :: operatörü yine choices içine bakılacağını söylüyor.

51.satırda std::map<string, int> yapısı kullanılarak olası ihtimaller ve karşılık gelen integer değerler map edilmiştir. Olası değerler burada ya left ya da right'dır. Fonksiyona başka bir choice string'i girdiğinde bu fonksiyonun çalışmaması exception atılması sağlanmalıdır.

58.satırda kontrol yapılmıştır choices_map.find(choice) ile verilen choice parametresi choices_map içerisinde aranacaktır find bir iterator döndürmektedir find iterator'ler aracılığı ile gidip choices_map içinde dolaşacak dolaşacak choices_map.end() iteratörüne denk gelirse if içerisine girecektir yani choices_map içerisinde tüm yerleri dolaştık fakat choice ile karşılaşamadık dolayısıyla fonksiyona verilen choice'un choices_map içerisinde yer almadığını anlamış olduk. Bu bağlamda if içerisine girersek return diyip fonksiyonu sonlandıracağız eğer programı sonlandırmak isteseydik exit(1) yazabilirdik fakat daha sonraki main içerisinde print_map() çağırımlarının çalışmasını isteyeceğimizden return demek daha doğru. (exit demek daha doğru olabilir sonuçta kodun bir satırında hata var print_map("rright") hata verir)

58.satıra alternatif olarak sayfa 9'da 5.1.5.1 Ekler başlığı altında EK 2'de alternatif basit bir çözüm yazılmıştır.

61.satırda map içerisinde key değerini kullanarak value değerine erişilmiş ve bu erişilen değer yeni bir choice_current integer değişkenine tutturulmuştur. Map içerisinde (choices_map) string ve int barındırıyordu string'ler left ve right olabiliyor bunlara karşılık gelen integer değerler enum yapısında belirlenmişti burada enum yapısının faydasını görebiliyoruz. Enum olmasa da olur muydu? Olurdu fakat manuel olarak map içine 0, 1 yazmamız gerekecekti ve aynı zamanda case'lere de 0 ve 1 yazmamız gerekecekti tam generic bir yapı olamayacaktı.

```
63      switch(choice_current)
64      {
65        case choices::left:
66          {
67            cout << "*** Iterating through the Map with the Left Key ***" << endl;
68            for (auto item = current_map.left.begin() ;
69                item != current_map.left.end();
70                ++item)
71                {
72                  cout  << " " << setw(10) << item->first << " -> "
73                        << setw(10) << (item->second)->get_number()
74                        << endl;
75                }
76          }
77
78          break;
79
80        case choices::right:
81          {
82            cout << "*** Iterating through the Map with the Right Key ***" << endl;
83            for (auto item = current_map.right.begin() ;
84                item != current_map.right.end();
85                ++item)
86                {
87                  cout  << " " << setw(10) << (item->first)->get_number() << " -> "
88                        << setw(10) << item->second
89                        << endl;
90                }
91          }
92          break;
93        default:
94          break;
95
96      } // switch
97    }  // function
98  } // namespace
99 } // namesapce
```

Artık switch içerisine choice'un karşılık geldiği integer değerler gönderilebilecektir.

63.satırda switch() statement'ina 0 değeri gelirse choice "left"dir, 1 değeri gelirse choice "right"dır.

65.satırda choices::left case'i aslında 0'ı ifade etmektedir. Choice current = 0 olursa left olan case gerçekleşecektir ve bu case içerisinde left key aracılığı ile iteratörler map içerisinde dolaşacaktır. 68.satırda for yapısı içerisinde value_type(string, smart pointer) current_map içerisinde bir pair vardır bu ikiliye first ve second keywordleri ile ulaşabiliyoruz. item->first ile key olan string bastırılacaktır, (item->second)->getnumber ile de bu map içerisinde objenin tuttuğu int number değeri bastırılacaktır.

İteratörler ile doğrudan bimap içerisinde dolaşmamız şarttır for(auto & item : current_map) yapısı çalışmayacaktır fakat map ve unordered_map yapılarında bu syntax çalışıyordu.

Burada right ve left'e bağlı olarak item->first ve item->second değişecektir çünkü;

Choice = left olan case'de key = string, value = smart_pointer'dı ve bu bağlamda bir ekrana bastırma yapısı tasarlandı. İtem->first ile string bastırılır.

Choice = right olan case'de key = smart_pointer, value = string olan bağlamda buna göre bir cout tasarlandı (item->first)->getnumber ile shared_ptr<Container> ' ın getter'i bastırılır.

Buradan anlaşılan o ki map ve unordered map yapısında map<,> solda kalan key oluyordu fakat bimap_type<,> yapısında solda olan bazen key bazen value olabilir.

```
101  int main()
102  {
103    using namespace std;
104    using namespace os_test;
105    using namespace os_test::common_typedefs;
106    using namespace os_test::utilities;
107
108    const int SZ = 5;
109    bimap_type container_map;
110
111    for(int kk=0; kk<5; ++kk)
112    {
113      std::stringstream ss;
114      char tmp_no[256];
115
116      sprintf(tmp_no, "%02d", 1+kk);
117      ss << tmp_no;
118
119      container_map.insert
120        ( value_type( "C"
121  //                      + std::string(tmp_no)
122                        + ss.str(),
123                      make_shared<Container>(10 * (SZ - kk))
124        ) );
125
126    }
127
128    print_map(container_map);
129    print_map(container_map, "right");
130
131
132    return 0;
133
134  }
135
```

Main içerisinde kullanılacak namespace'ler çağırılmıştır, const SIZE belirlenmiş 5 olarak seçilmiş yani map içerisinde toplam 5 tane pair olacaktır. Bimap_type typedef'i kullanılarak bir string, smart pointer ikilisini tutacak bir map değişkeni oluşturulmuştur isim olarak container_map seçilmiş.

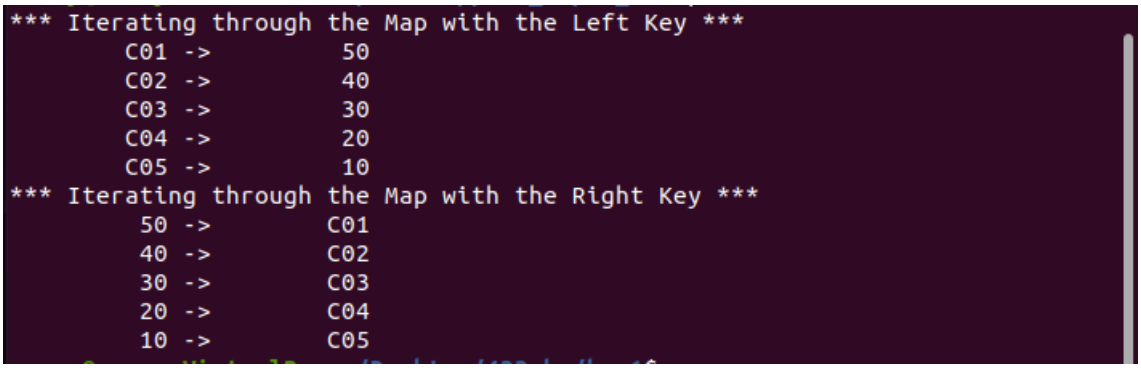
111.satırda for döngüsü içerisinde sprintf()'in kullanılma amacı tmp_no içerisine 1+kk'nin %02d formatında yazılması söz konusudur tmp_no aslında bir birbiri ardına sıralanmış hafıza bloklarıdır burada [256] seçilmiştir 255 karakter + 1 NULL karakterini barındırabilecektir. Kk = 0 olduğu durum için bakarsak tmp_no içerisine 01 yazılacaktır, bir sonraki satırda ss << tmp_no diyerek ostream operatörü kullanılarak stringstream içerisine tmp_no atılmaktadır.

119.satırda yukarıda tanımlanan bimap_type container map'e pairleri atıyoruz vector.push_back'e benzemektedir, map'in en son alanına iteratif olarak pairleri yerleştirecektir.

120.satırda value_type(string, smart pointer) olması gerekmektedir çünkü typedef bu şekilde belirlendi, "C" + ss.str() yapılabilir çünkü soldaki "C" const char *'dır sağdaki ss.str() ise bir std::string'dir bu iki yapı için "+" operatörünün C++'da overload edildiği bilinmektedir. İkinci olarak make_shared<Container>(10 * (SZ - kk)) bir smart pointer'ı oluşturarak buna ilk değer ataması yapmaktadır (Perfect Forwarding) 5 kere bu işlem yapılmıştır.

Son olarak utilities içerisinde yazılan print_map fonksiyonu kullanılarak output'un ekrana bastırılması sağlanmıştır.

Output:



Beklendiği gibi key, value pair üzerinden bu ikilinin birbirlerinin yerine geçebildiği gözlemlenmiştir.

5.1.5.1 için EKLER:

EK 1:

In C++:

```cpp
#include<iostream>
#include<cstring>
#include<string>
#include<iomanip>
#include<memory>
#include<sstream>
#include<map>
#include<unordered_map>
#include<algorithm>
#include<vector>

using namespace std;

namespace all_tests{

void unordered_map_test()
{
  cout << "**** UNORDERED_MAP_TEST **** "<< endl;

  using tuple_type
    = std::tuple<int,int,int>;

  auto red_tuple = std::make_tuple(255, 0, 0);
  auto green_tuple = std::make_tuple(0, 255, 0);
  auto blue_tuple = std::make_tuple(0, 0, 255);

  using unordered_map_type
    = std::unordered_map<std::string, tuple_type>;

  unordered_map_type
    t_map =
      {
        {"RED"  , red_tuple},
        {"GREEN", green_tuple},
        {"BLUE" , blue_tuple},
      };
```

```cpp
  auto fh_print
    = [](unordered_map_type & map_input)->void
    {
      cout << "** fh_print **" << endl;
      for(auto &item : map_input)
      {
        cout << " Color String : " << item.first << setw(20)
             << endl << "    Pixel value  : " << "( "
             << std::get<0>(item.second) << ", "
             << std::get<1>(item.second) << ", "
             << std::get<2>(item.second) << ") "
             << endl;
      }
    };

  fh_print(t_map);
  cout << endl;

} // function
```

C++'da unordered_map veya map yapısı kullanılarak key, value çiftinde key kullanılarak value değiştirilebilir fakat value değeri kullanılarak key değeri değiştirilemez.

Unordered_map_test() fonksiyonu içerisinde basit bir color space'de kırmızı, yeşil, mavi renklere karşılık gelen pixel değerlerini string ve pixel değeri çifti oluşturularak bir map yapısı yapılmıştır. Amaçlanan şey aslında unordered_map yapısını tekrar hatırlamaktır. Bu testin çıktısı aşağıda gösterilmiştir.

```
soray@soray-VirtualBox:~/Desktop/433_hw/hw_1$ g++ map_types.cpp -o test1.out
soray@soray-VirtualBox:~/Desktop/433_hw/hw_1$ ./test1.out
**** UNORDERED_MAP_TEST ****
** fh_print **
 Color String : BLUE
     Pixel value  : ( 0, 0, 255)
 Color String : GREEN
     Pixel value  : ( 0, 255, 0)
 Color String : RED
     Pixel value  : ( 255, 0, 0)
```

```cpp
void map_test()
{
  cout << "**** MAP_TEST **** "<< endl;
  using tuple_type
    = std::tuple<int,int,int>;

  auto red_tuple = std::make_tuple(255, 0, 0);
  auto green_tuple = std::make_tuple(0, 255, 0);
  auto blue_tuple = std::make_tuple(0, 0, 255);

  using map_type
    = std::map<std::string, tuple_type>;

```

```cpp

  map_type
    t_map =
      {
        {"RED"  , red_tuple},
        {"GREEN", green_tuple},
        {"BLUE" , blue_tuple},
      };

  auto fh_print
    = [](map_type & map_input)->void
    {
      cout << "** fh_print **" << endl;
      for(auto &item : map_input)
      {
        cout << " Color String : " << item.first << setw(20)
             << endl << "    Pixel value  : " << "( "
             << std::get<0>(item.second) << ", "
             << std::get<1>(item.second) << ", "
             << std::get<2>(item.second) << ") "
             << endl;
      }
    };

  fh_print(t_map);
  cout << endl;
```

map_test() testi yukarıdaki örneğe benzer bir testtir fakat bununla beraber bazı şeyler gözlemlenmiştir. Aynı yapı basitçe tekrar kurulmuştur, using keyword'u burada işimizi birkaç yerde kolaylaştırmıştır, birden fazla integer değeri tutmak için tuple yapısı kullanıldı ve fh_print() lambda fonksiyonu da basitçe çıktıyı görmemizi sağladı.

```cpp
  t_map["RED"] = std::make_tuple(0, 0, 0);
  fh_print(t_map);

  // HERE I want to take blue's 255 int value.
  map_type::iterator it;
  it = t_map.begin();
  cout << endl;
  cout << "BLUE's 255 value : " << std::get<2>(it->second) << endl;
  cout << endl;

//   t_map[red_tuple] = "R_RED" ;
//   It is not possible
```

Yapı map<string, tuple> şeklinde oluşturulduğu için burada key string olmaktadır herhangi bir string değeri kullanılarak value değiştirilmesi amaçlanmıştır (97.satır). Value değiştirilip tekrar t_map gözlemlenmiştir değişebildiği görülmüştür fakat 107.satırda value değeri kullanılarak key değiştirilmeye çalışıldığında bu işlemin işlemediği görülmüştür.

```cpp

//   SAME SECOND BASIC EXAMPLE
  cout << "  Second Example  " << endl;
  std::map<int,std::string>
    m =
    {
      {1, "ONE"},
      {2, "TWO"},
    };

  auto fh_print_new
    = [](std::map<int, std::string> map_input)->void
    {
      cout << "** fh_print_new **" << endl;
      for(auto &item : map_input)
      {
        cout << "Int value    : " << item.first << " -> "
             << "String value : " << item.second << endl;
      }
      cout << endl;
    };

```

```
132     fh_print_new(m);
133
134     m[1] = "BIR";
135     m[2] = "IKI";
136
137     fh_print_new(m);
138
139   //   It is not possible
140   //   m["BIR"] = 11;
141   //   m["IKI"] = 22;
142
143   //   fh_print_new(m);
144
145   } // function
146
147   } // namesapce
```

Daha basit bir örnekle map<int, string> yapısı ile de aynı işlemler denendi 140 ve 141. satırların işletilemeyeceği gözlemlenmiştir. map_test() testinin outputu aşağıda gösterilmiştir.

```
soray@soray-VirtualBox:~/Desktop/433_hw/hw_1$ g++ map_types.cpp -o test2.out
soray@soray-VirtualBox:~/Desktop/433_hw/hw_1$ ./test2.out
**** MAP_TEST ****
** fh_print **
 Color String : BLUE
     Pixel value  : ( 0, 0, 255)
 Color String : GREEN
     Pixel value  : ( 0, 255, 0)
 Color String : RED
     Pixel value  : ( 255, 0, 0)

** fh_print **
 Color String : BLUE
     Pixel value  : ( 0, 0, 255)
 Color String : GREEN
     Pixel value  : ( 0, 255, 0)
 Color String : RED
     Pixel value  : ( 0, 0, 0)

BLUE's 255 value : 255

  Second Example
** fh_print_new **
Int value    : 1 -> String value : ONE
Int value    : 2 -> String value : TWO

** fh_print_new **
Int value    : 1 -> String value : BIR
Int value    : 2 -> String value : IKI
```

```
149 int main()
150 {
151   using namespace std;
152   using namespace all_tests;
153
154   unordered_map_test();
155   map_test();
156
157   return 0;
158 }
```

unoredered_map ile map arasındaki farklar:

map yapısı içerisinde key değerleri sıralı bir dizi olacaktı yani 116 ve 117.satırların yeri değişmiş olsa dahi çıktı aynı kalacaktı fakat unordered_map' de çıktı aynı kalmayacaktı çıktıda önce 2 ardından 1 için bir çıkış gözlemlenecekti.

Unordered_map ve Map içerisinde aynı şeyler olsun bu yapılar içinde bir şey aramak istersek map yapısı daha hızlı bulacaktır binary search ile linear search arasındaki farka benzemektedir.

EK 2:

```
59      /*
60      if( choices_map.find(choice) == choices_map.end() )
61      {
62        throw std::invalid_argument
63          ("Can not such a choice");
64      }
65      */
66      try
67      {
68        if( choices_map.find(choice) == choices_map.end() )
69          throw choice;
70      }
71
72      catch(std::string choice)
73      {
74        cout << " choice parameter can not be such " << choice << " choice " << endl;
75        return;
76      }
```

try catch yapısı kullanarak execption atılmıştır catch ile yakalandıktan sonra bilgilendirme mesajı ekrana bastırılacaktır ve fonksiyondan çıkılacaktır.

```
144
145   print_map(container_map, "rightw");
146   print_map(container_map);
147   print_map(container_map, "rright");
148
149
```

Burada olduğu gibi yanlış choice parametreleri verilebilir bu durumlarda fonksiyonun çalışmaması sağlanır diğer durumlarda fonksiyon çalışacaktır. Programın hiç çalışmaması istenirse ki bunun istenmesi daha olası 59 ile 65. Satırlar arasında yorum satırına alınan yer comment out yapılmalıdır. Try catch yapısı da comment in yapılması gerekir. Exception atılacak fakat tutulmayacaktır dolayısı ile program sonlanacaktır. Exit(1) kullanılması daha doğru olabilirdi.

try catch output:

```
soray@soray-VirtualBox:~/Desktop/433_hw/hw_1$ ./a.out
 choice parameter can not be such rightw choice
*** Iterating through the Map with the Left Key ***
        C01 ->          50
        C02 ->          40
        C03 ->          30
        C04 ->          20
        C05 ->          10
 choice parameter can not be such rright choice
```

std::invalid arguments output:

```
terminate called after throwing an instance of 'std::invalid_argument'
  what():  Can not such a choice
Aborted (core dumped)
```

5.1.5.2 initial_boost_bimap_example_better.cc

In C++:

```cpp
#include <sstream>
#include <map>
#include <type_traits>

#include <boost/bimap.hpp>

namespace os_test {

  class Container
  {
  public:
    Container
    (int number = 0)
    :
    number_(number)
    {}

    ~Container() = default;

    int
    get_number
    ()
    {
      return number_;
    }

    friend
    std::ostream& operator<<(std::ostream & os, const Container & c);

  private:
    int number_;
  };

  std::ostream& operator<<
  (std::ostream & os, const Container & c)
  {
    os << c.number_;
    return os;
  }
```

Aynı problemi daha iyi daha generic bir yöntem ile çözülebilmesi için bu kod yazılmıştır, 5.1.5.1 kodunun switch yapısı içerisinde “code reuse” yapılıyordu, bundan kurtulmak amaçlanmıştır. Fakat bu amaç beraberinde birçok yapıyı gerektirecektir.

Encapsulation için namespace açıldı ve aynı şekilde bir Container class yazıldı bunun dışında bu class’ın objeleri için bir ostream overloading’i yapılmıştır. Neden friend olmak zorunda?

Friends’ler public içinde fakat global fonksiyonlar gibi davranırlar. Global fonksiyonları sınıf içerisine sokuyoruz ve sınıfın private memberlarına ulaşabiliyoruz, gidip de getter’larla uğraşmaya gerek kalmıyor. Neyi yazdırmak istiyorsak os << ardından yazdırmak istediğimiz şeyi yazmalıyız. En sonda bu ostream operatörünü return etmeliyiz ki class’ın objeleri için artık bir anlam kazanabilsin.

```cpp
  namespace common_typedefs {
    using bimap_type
      = boost::bimap<std::string, os_test::Container*>;
    using value_type
      = bimap_type::value_type;
    using map_of_pointers_type
      = std::map
        <os_test::Container*, std::shared_ptr<os_test::Container>>;

    using tuple_of_maps_type
      = std::tuple<bimap_type, map_of_pointers_type>;
  } // namespace
```

56.satırda typedeflerin yapıldığı namespace yer almaktadır bunun içinde sırası ile bimap_type(string, Container classın objesi olacak raw pointer), value_type bir önceki koddaki ile aynı işi yapacaktır, map_of_pointers_type(raw pointer, smart pointer), tuple_of_maps_type(bimap_type, map_of_pointers_type) olarak typedef’ler yapılmıştır burada map_of_pointers_type’ın görevi smart pointer’ın içerisindeki raw pointer’ı smart pointer ile aynı mapa sokmaktır. Tuple_of_maps_type ise bimap içerisindeki raw pointer ile bu map_of_pointer_type içerisinde saklanan raw pointer’ı aynı olacaktır.

Dolaylı yoldan bir string ile smart pointer’ın içindeki raw pointer map edilmiş oluyor.

```
69  namespace utilities {
70    using namespace os_test::common_typedefs;
71
72    template
73    < typename T,
74      typename
75        std::enable_if<
76          std::is_pointer<T>{},
77          int>::type = 0 >
78    std::string
79    printer_auxiliary
80    (T item)
81    {
82      std::stringstream ss;
83      ss << *item;
84      return ss.str();
85    }
```

```
87    template
88    < typename T,
89      typename
90        std::enable_if<
91          !std::is_pointer<T>{},
92          int>::type = 0 >
93    std::string
94    printer_auxiliary
95    (T item)
96    {
97      return item;
98    }
```

Bu iki template fonksiyona beraber bakılmalıdır, overload edilmiş iki template yapısı gözükmektedir, enable_if yapısı burada bazı şartlar oluşturarak iki fonksiyonu birbirinden ayırmıştır.

70.satırda yine kullanılacak typedef'ler buradaki utilities nested namespace'ine çağırıldı.

72.satırdaki template fonksiyonun çağırılabilmesi (run time'da oluşturulup kullanılabilmesi için ne gerekir?) printer_auxiliary<pointer> şeklinde bir çağırım gerçekleşir ise bu fonksiyon oluşturulacaktır.

87.satırdaki template ise print_auxiliary<not pointer> çağırımı pointer olmayan bir şey ile yapılırsa bu fonksiyon oluşturulup kullanılabilecektir. Bu 2 template'e ayrı ayrı neden ihtiyaç var?

Çünkü bir önceki kodda 5.1.5.1'de switch içerisinde for döngüleri içinde left case için item->first, right case için ise (item->first)->get_number yazıyorduk bunu ortadan kaldırmak için her iki durum içinde (bu her iki durum şudur; string'in key olduğu veya smart_pointer'ın key olduğu durumlar) item->first yazabilmek için bunu yapmak içindir ki, yukarıda ostream(<<) overload edildi.

72.satırdaki template'e raw pointer geliyor onu dereference edip içindeki değer ne ise onu stringstream içine atıyoruz içindeki değere nasıl ulaşabiliyoruz içindeki private bir member (<<) operatörü yardımcı oluyor.

87.satırdaki template'e pointer olmayan bir şey yani string gelmek zorunda burada da yapılan tek iş gelen string'i return etmek.

```
100   template
101   <typename T>
102   std::function<void(std::string, const T &)>
103   get_printer_lambda
104   ()
105   {
106     using namespace std;
107     auto fh_print =
108       []( std::string id,
109           const T &   internal_map)->void
110         {
111           cout
112             << "*** Iterating through the Map with the "
113             << setw(5) << id << " Key ***"
114             << endl;
115           for (
116             auto
117             item  = internal_map.begin() ;
118             item != internal_map.end() ;
119             ++item
120           )
121           {
122             cout
123               << "  " << setw(10)
124               << printer_auxiliary(item->first) << " -> "
125               << setw(10)
126               << printer_auxiliary(item->second)
```

```
126                 << printer_auxiliary(item->second)
127                 << endl;
128             }
129           };
130       return fh_print;
131     }
```

100.satırda başlayan get_printer_lambda template fonksiyonu bir fonksiyon return edecektir bu fonksiyonun lambda fonksiyon olduğu ve Lambda'nın kendisinin void return ettiği ve input parametreleri olarak ise string ve yollanacak diğer parametrenin tipi olacaktır.

Bu yollanacak diğer parametrenin tipi muhtemelen bimap_type.left veya bimap_type.right olmalıdır. Lambda içerisinde gelen tipe göre bir yapı kurulmuş printer_auxiliary() fonksiyonları burada işe yarıyor gelen tipin elemanlarının yani maplerin ilk item'ı (item->first) eğer string ise 87.satırdaki template çalışacak string return edilecek ve ardından bastırılacak eğer ilk item pointer ise 72.satırdaki template çalışacak ve bir stringstream return edilecektir (stringstream içerisinde bir number var) class'ın objesine << operatörü aracılığı ile private değeri bastırılabiliyor.

İtem->first için yapılan şeyler item->second içinde yapılıyor bunun içinde aynı kontroller yapılıyor hangi print_auxilariy'e girmesi gerekiyorsa giriyor ve sonuç ekrana bastırılıyor.

Sonuç olarak get_print_lambda içerisinde bir lambda fonksiyon return ediliyor, neden böyle bir yol izlenmiş bir sonraki bölümde değinilecektir.

```
133   void
134   print_map
135   (
136     const tuple_of_maps_type & current_tuple ,
137     std::string                choice = "left"
138   )
139   {
140     using namespace std;
141     const auto & current_map = std::get<0>(current_tuple);
142
143     enum choices
144     {
145       left  = 0 ,
146       right
147     };
148
149     std::map<std::string, int>
150       choices_map =
151         { {"left" , choices::left  } ,
152           {"right", choices::right } };
153
154     // NOTE: Necessary to check if "choice" is in "choices_map".
155     // TODO: Can/Should throw an exception if key is not found.
156     if ( choices_map.find(choice) == choices_map.end() )
157       return;
158
```

133.satırda yazılan print map fonksiyonu input parametresi olarak bir tuple alıyor bu tuple gönderilirken const referans olarak yani read only şekilde gönderiliyor kopyalama yok (pass by ref.) choice kısmı 5.1.5.1'deki ile aynıdır. Enum yapısı ve ardından gelen map yapısı yine önceki script'deki gibi kullanılmış bir if yapısı ile bu choice'un istenmeyen yani right veya left dışında olduğu durumda fonksiyondan çıkılması söylenmiştir.

```
159     int choice_current = choices_map[choice];
160     switch( choice_current )
161     {
162       case choices::left:
163         { // scope - begin
164           using internal_map_type
165             = decltype(current_map.left);
166           auto fh_print
167             = get_printer_lambda<internal_map_type>();
168           fh_print("Left" , current_map.left );
169         } // scope - end
170         break;
171       case choices::right:
172         { // scope - begin
173           using internal_map_type
174             = decltype(current_map.right);
175           auto fh_print
176             = get_printer_lambda<internal_map_type>();
177           fh_print("Right", current_map.right);
178         } // scope - end
179         break;
180       default: // not necessary
181         break;
```

Gönderilen tuple ne tutuyordu? (Bimap_type içerisinde string ve raw pointer ikilisi) ve (raw pointer ve o raw pointer'ın yönetimini üstlenen smart pointer ikilisi) current_map = get<0>(tuple) string ve raw pointer ikilisini tutmaktadır yani bimap_type'ı tutmaktadır. Bu ikili tuple içerisinden std::get ile çekilmiştir.

162.satır yani case choices::left yani case 0 olduğu durumda; decltype(current_map.left) decltype burada using internal_map_type yapısı ile bir type kazanımı sağlamıştır yani internal_map_type bir tiptir ve curent_map.left tipindedir. İstenilen durum da budur buradaki internal_map_type'ı 100.satırda yazılan get_print_lambda'ya gönderilecektir. Get_print_lambda left case'i için bir ekrana yazdırma yani string'in key olduğu raw pointer'ın value olduğu durumda bir lambda fonksiyonu return edecektir.

166.satırda return edilen lambda'yı tutan bir fh_print() tanımlaması yapılmıştır return edilen bu lambda fonksiyonu left case için geçerlidir ve bu case'de istenilen çıktılar verilecektir. Fh_print(string, current_map.left) almaktadır.

171.satırda right case için left case'e benzer işlemler yapılmıştır bu case için declltype tuple içerisinde gelen map'i özelleştirerek yeni bir map tanımı yapmıştır (daha doğrusu map içerisindeki sağda bulunan elemanı key olarak değerlendirmiştir sağdaki eleman burada pointer dolayısı ile right case'inde output'da sol'da yani key'in olması bastırılması gereken yerde sayılar olacaktır neden solda olmalı? çünkü key -> value key value'i işaret etmeli)

```
182       } // switch
183     } // print_map
184   } // namespace
185 } // namespace
```

```
187 namespace {
188   using namespace std;
189   using namespace os_test;
190   using namespace os_test::common_typedefs;
191   using namespace os_test::utilities;
192
193   tuple_of_maps_type
194   create_data
195   ()
196   {
197     const int SZ = 5;
198     bimap_type
199       container_map;
200     map_of_pointers_type
201       pointers_map;
202
203     for ( int kk=0 ; kk<5 ; ++kk )
204     {
205       std::stringstream ss;
206       char tmp_no[256];
207
208       std::shared_ptr<Container> tmp_ptr
209         = make_shared<Container>(10 * (SZ-kk));
210
211       sprintf( tmp_no , "%02d" , 1+kk );
212       ss << tmp_no;
213       pointers_map[&(*tmp_ptr)] = tmp_ptr;
214       container_map.insert
215         ( value_type( "C" + ss.str() ,
216                       &(*tmp_ptr)
217         ) );
218     }
219
220     tuple_of_maps_type
221       maps_tuple(container_map, pointers_map);
222     return maps_tuple;
223   }
224 } // namespace
225
```

Burada namespace altında bir tuple return eden create_data() fonksiyonu yazılmıştır. Bu fonksiyon içerisinde gerçekleşen şeyler sırası ile;

197.satırda bir SIZE belirlendi bu size map içerisinde kaç tane pair olması gerektiğini belirler.

198 ve 200.satırlarda bimap_type (string, raw ptr) ve map_of_pointers_type (raw ptr, shared ptr) typedef'leri ile değişkenler oluşturuldu.

203.satırdan itibaren for içerisinde sprintf() 5.1.5.1 'deki gibi kullanılıyor, amaç tmp_no'ya belli formatta bir şeyler atamak daha sonra bu tmp_no stringstream'e yollayacağız.

208.satırda bir smart_pointer perfect forwarding tekniği ile make_shared sayesinde bir ilk değer kazanıyor. tmp_ptr bir Container smart pointer instance'dir, içerisinde bir ham işaretçi ve bu ham işaretçinin otomatik öldürülebilmesi için referans sayma mekanizması vardır.

213.satırda oluşturulan pointers_map 'e insert işlemi yapılıyor yani son ikili konumuna bir atama yapılıyor bu atamada key olarak 208.satırda oluşturulan tmp_ptr smart pointer'ının yönettiği ham işaretçisi çekip alınıyor bunu alırken &[*tmp_ptr] kullanılmış get ile de alınabilirdi ve bu alınan ham işaretçiye pair olarak yani value olarak smart pointer'ın kendisi atanmıştır dolayısı ile ham işaretçi kullanılarak smart pointer'a erişim sağlanabilmektedir.

pointers_map artık bir (smart pointer'ın içindeki raw pointer, smart pointer'ın kendisi) ikilisini tutmaktadır.

214.satırda artık bimap_type container_map'a pair'ler insert edilecektir. Bu bidirectional map'e pair çifti olarak string ve 208.satırda oluşturulan ve pointers_map'ın da barındırdığı smart pointer'ın içindeki raw pointer verilmiştir. Dolaylı yoldan bir string ve bir smart pointer pair olarak map içerisine sıkıştırılmıştır.

Sonuç olarak bidirectional map ve map_of_ptr olarak iki ayrı map bir tuple'a tutturulmuş ve bu tuple return edilmiştir. Kodda sadece bu tuple içindeki bidirectional_map kısmı kullanılmıştır.

```
226  int
227  main (void)
228  {
229    using namespace std;
230    using namespace os_test;
231    using namespace os_test::common_typedefs;
232    using namespace os_test::utilities;
233
234    auto maps_tuple = create_data();
235
236    print_map(maps_tuple);
237    print_map(maps_tuple, "right");
238
239    return 0;
240  }
241
```

Output:

```
soray@soray-VirtualBox:~/Desktop/433_hw/hw_1$ g++ initial_boost_bimap_exampl
e_better.cpp
soray@soray-VirtualBox:~/Desktop/433_hw/hw_1$ ./a.out
*** Iterating through the Map with the  Left Key ***
         C01 ->          50
         C02 ->          40
         C03 ->          30
         C04 ->          20
         C05 ->          10
*** Iterating through the Map with the Right Key ***
          50 ->         C01
          40 ->         C02
          30 ->         C03
          20 ->         C04
          10 ->         C05
```

Ilk olarak left case gerçeklenmiştir left case için kodun üzerinden geçmek gerekirse get_print_lambda içerisinde print_auxilary(item-first) için 87.satıdaki template işlemektedir çünkü key olarak string kullanılıyor. print_auxilary(item->second) için 72.satır pointer'ın template fonksiyona girdiği durum kullanılmaktadır. Second olarak ptr var çünkü bu case'de ptr bir value'dir.

Print_map() fonksiyonu içerisinde switch yapısında case : Left'e girecektir ve internal_type olarak current_map yani bimap_type.left oluşturulacak ve get_printer_lambda'ya gönderilecek bu tip üzerinden

bir lambda fonksiyonu oluşturulacak ve return edilecektir. Return edilen bu lambda ilgili giriş parametrelerini alarak kullanılacaktır. Bu case'de sol tarafta string sağ tarafta ise ptr gözlemlenecektir.

Key->Value olacağından String->PTR beklenmektedir.