

LAPORAN TUGAS BESAR I

IF2210 PEMROGRAMAN BERORIENTASI OBJEK



Disusun oleh :

Kelompok 07 – Monangisbeneran – K01

13520004 / Gede Prasadha Bhawarnawa

13520064 / Ziyad Dhia Rafi

13520133 / Jevant Jedidia Augustine

13520136 / Vincent Christian Siregar

13520160 / Willy Wilsen

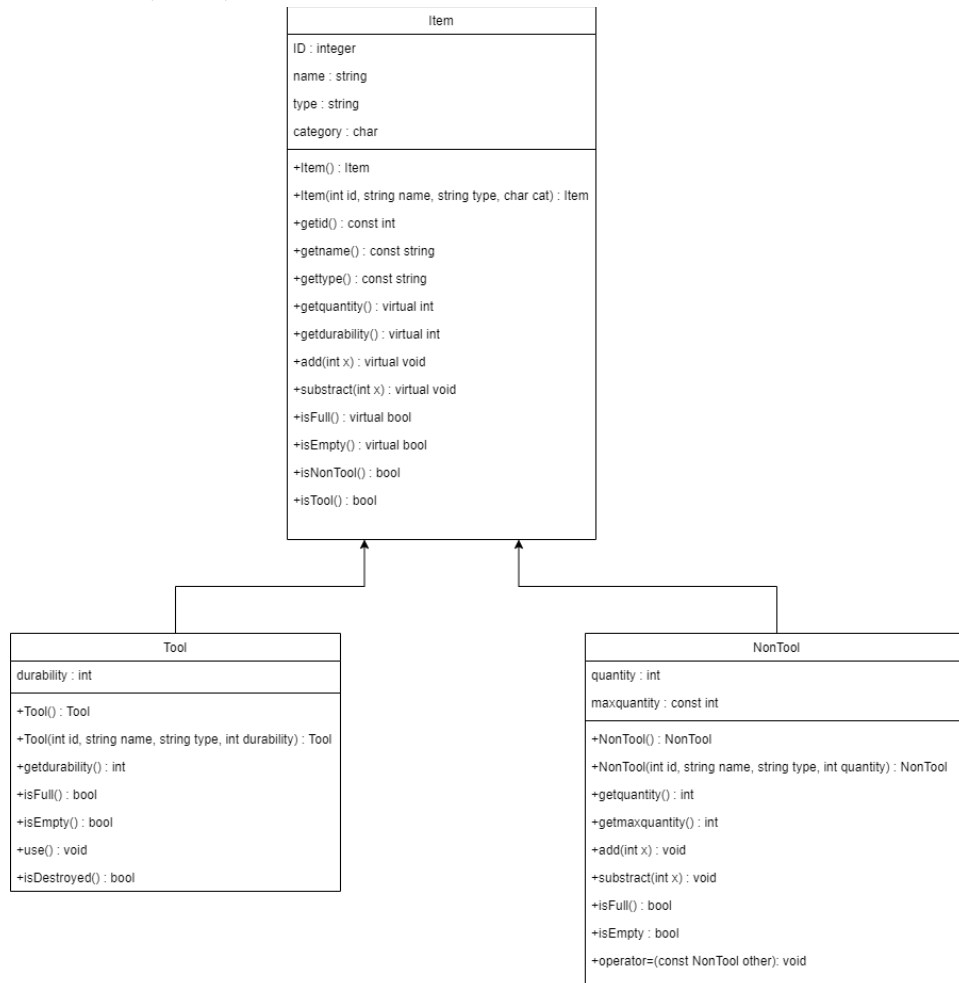
SEKOLAH TEKNIK ELEKTRO DAN INFORMATIKA

INSTITUT TEKNOLOGI BANDUNG

2022

1. Diagram Kelas

A. Kelas Item, Tool, dan NonTool



Alasan penulis mendesain struktur kelas untuk Item, NonTool, dan Tool seperti ini adalah penulis berpendapat bahwa NonTool dan Tool berbagi banyak kesamaan atribut, seperti id, nama, dan tipe. Perbedaan utama dari NonTool dan Tool terletak pada saat diletakkan di dalam slot Inventory atau Crafting Table. Satu slot pada Inventory atau Crafting Table dapat ditempati oleh maksimum 64 buah NonTool yang sama. Sementara itu, jika ingin ditempati oleh Tool, satu slot hanya dapat menampung maksimum 1 buah. Selain itu, pada Tool juga terdapat atribut durability yang tidak terdapat pada NonTool. Maka dari itu, penulis memutuskan akan lebih modular dan lebih mudah dipahami, dibaca, dan diimplementasikan bila dibuat inheritance terhadap Tool dan NonTool dengan Item sebagai *parent class*-nya.

B. Kelas Recipe dan RecipeList

Recipe
RecipeContents : array of string of string
RecipeResult : string
resultQty : int
rowSize : int
colSize : int
maxRowSize : const int
maxColumnSize : const int
+Recipe() : Recipe +setRecipeResult(string newResult) : void +getRecipeResult() : const string +setResultQty(int newQty) : void +getResultQty() : const int +setRowSize(int rowSize) : void +getRowSize() : int +setColSize(int colSize) : void +getColSize() : int +getIngredientByLocation(int row, int column) : const string +insertIngredient(int row, int column, string ingredient); +printRecipe() : void +checkRecipe(Table<3,3>* c); +mirrorY() : Recipe

RecipeList
List : vector<Recipe>
+RecipeList() : RecipeList +addRecipes(const Recipe newRecipe) : void +checkPurePairTool(Table<3,3>* C) : bool +checkCrafting(Table<3,3>* C) : Recipe +operator<<(const string& folder) : void +getFilesInFolder(string folder) : vector<filesystem::path> +splitString(string full) : vector<string> +showAll() : void

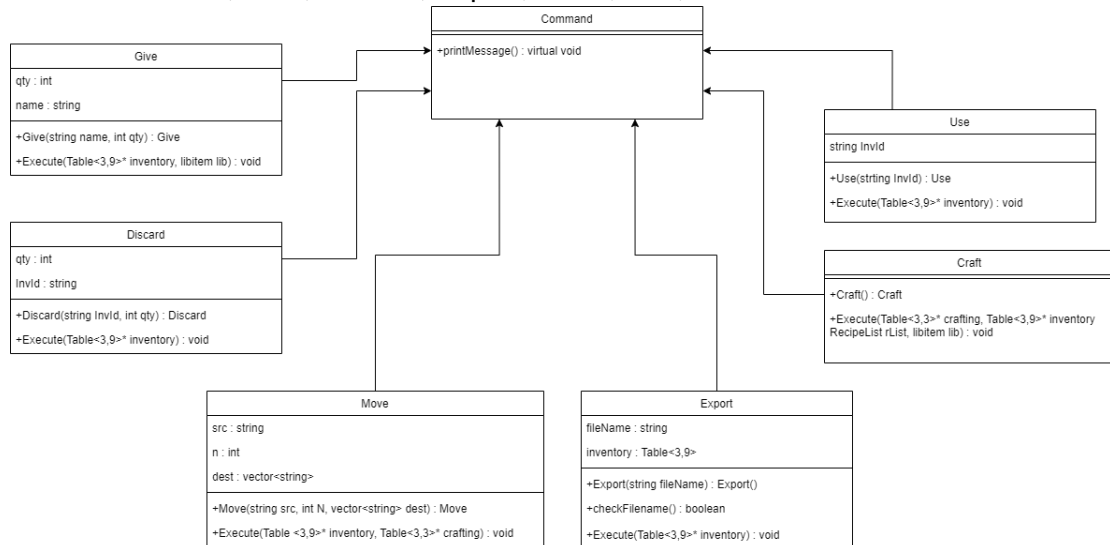
Kelas Recipe adalah kelas yang digunakan untuk menyimpan resep-resep crafting pada program. Pada awalnya terdapat kesulitan dalam menentukan tipe data untuk menyimpan informasi ukuran Recipe, serta elemen-elemen yang dibutuhkan untuk menghasilkan produk dari Recipe. Beberapa pertimbangan yang sempat dicoba oleh penulis adalah penggunaan STL vector, STL array, dan default C++ static array. Setelah beberapa percobaan, disepakati oleh tim penulis bahwa bentuk penyimpanan detail Recipe akan disimpan dalam bentuk array of string of string, atau array berdimensi dua berisikan string. Masing-masing string akan menyimpan detail dari komponen Recipe dan untuk mengakses masing-masing komponen digunakan indexing array zero-based. Selain array of string of string, digunakan juga tipe data string untuk menyimpan hasil dari Recipe dan pasangan tipe data integer untuk menyimpan ukuran dari Recipe itu sendiri.

Selanjutnya, mengingat ada beberapa Recipe yang diimplementasi pada program ini, Recipe-Recipe tersebut perlu disimpan di dalam sebuah struktur data. Untuk menjawab persoalan ini, penulis memanfaatkan kelas RecipeList untuk menyimpan Recipe tersebut dengan menggunakan STL vector. Sama seperti list biasa, Recipe pada RecipeList dapat diakses menggunakan indexing zero-based dan dimanipulasi menggunakan fungsi, termasuk salah satunya berupa operator overloading.

C. Kelas *Table*

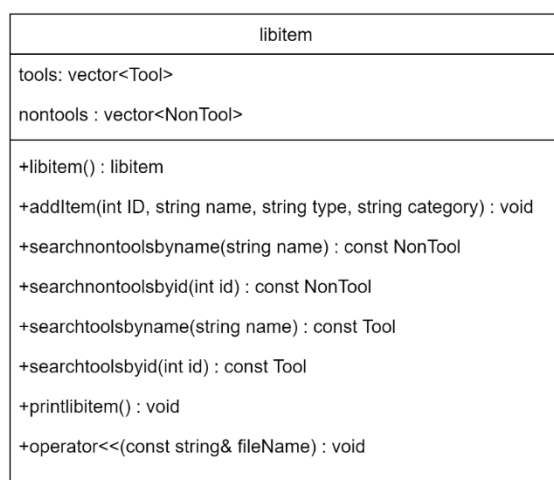
Implementasi kelas *Table*, dibahas lebih lanjut pada bab 2 (Penerapan Konsep OOP), didesain oleh penulis untuk menggunakan *generic class*. Alasannya adalah terdapat dua buah *instance* yang dibangun dengan kelas *Table* namun memiliki implementasi pada fungsi utama yang berbeda. Kedua instance ini adalah *Inventory* dan *Craft* (*Crafting Table*). Sebelumnya dipertimbangkan pemisahan menjadi dua kelas yang berbeda, satu untuk setiap *instance*. Namun, penulis menyadari banyaknya kesamaan atribut dan fungsi-fungsi dasar antara kedua objek tersebut. Selain itu, dengan mengimplementasikan *template*, maka dapat dibuat spesifik sebuah fungsi agar hanya dapat menerima *Craft* atau *Inventory* saja.

D. Kelas Command, Give, Discard, Export, Move, Use, Craft



Pada diagram kelas diatas, penulis telah merancang command diimplementasikan menggunakan inheritance. Alasannya adalah pada program ini terdapat beberapa command yang sifatnya memodifikasi isi dari list yang telah di-*construct* sebelumnya, seperti GIVE, DISCARD, dan USE. Agar memudahkan bagi pengguna untuk mengetahui apakah commandnya berhasil dilaksanakan atau tidak, akan dioutputkan pesan melalui CLI. Pada prinsipnya, proses mengeluarkan pesan untuk setiap command sama. Perbedaannya terletak pada isi pesannya. Agar mengurangi jumlah kode yang ditulis dan meningkatkan modularitas, dibuatlah semua command sebagai *child* dari satu buah *parent class* bernama "Command". Pada kelas Command, tidak ada atribut private, melainkan hanya ada satu buah *pure virtual function* yang akan dijabarkan implementasinya pada kelas childnya. Penggunaan diagram kelas diatas menyelesaikan permasalahan repetisi penggunaan kode "cout" dan membuat kode lebih bersih dan modular.

E. Kelas LibItem



Kelas LibItem didesain oleh penulis agar dapat menyimpan semua Item (baik itu Tool maupun NonTool) yang sudah didaftarkan ke dalam program saat pertama kali diaktifkan. Dengan ini, pengguna mendapat batasan item-item yang dapat mereka tambahkan ke dalam inventory mereka dan bersama dengan itu juga batasan crafting

mereka. Desain ini menggunakan dua buah vector of Item: vector of Tool dan vector of NonTool. Ada dua keuntungan yang didapatkan dengan memisahkan penyimpanan Item ke dalam dua kategori meskipun secara teori hasil yang sama tetap didapatkan jika menggunakan satu buah vector of Item karena Tool dan NonTool adalah hasil inheritance dari Item. Keuntungan pertama adalah ini memudahkan proses command GIVE, atau command menambahkan Tool dan/atau NonTool ke dalam inventory. Alasannya adalah fungsi cukup memeriksa Item berada di vector of Tool atau vector of NonTool sesuai tipe Item yang diperiksa pertama kali. Secara efisiensi waktu, mengasumsikan jumlah isi vector of Tool dan vector of NonTool sama, ini menghemat waktu hingga setengahnya. Keuntungan kedua adalah memudahkan pencarian Recipe saat dilakukan crafting. Setelah mendapatkan hasil Recipe sesuai dengan komponen yang terdapat pada Craft, program membutuhkan detail lebih lengkap agar hasil crafting dapat ditambahkan ke dalam inventory pengguna. Detail ini dapat dicari pada libitem lebih cepat dengan alasan yang sama pada keuntungan pertama.

2. Penerapan Konsep OOP

2.1. Inheritance & Polymorphism

Konsep inheritance digunakan pada kelas Item, Tool, dan NonTool. Kelas item adalah kelas yang menyimpan detail dari sebuah item yang terdapat pada game. Kelas NonTool merupakan kelas anak dari kelas Item yang menyimpan informasi quantity dan maxquantity yang menyatakan jumlah NonTool yang terdapat pada 1 objek. Kelas Tool merupakan kelas anak dari kelas Item yang menyimpan informasi durability yang menyatakan ketahanan dari sebuah Tool.

```

#ifndef _ITEM_HPP_
#define _ITEM_HPP_

#include <iostream>
#include <string>

using namespace std;

class Item {
private:
    const int id;
    string name;
    string type;
    char category; //T for tool, N for nontool
public:
    Item();
    Item(int id, string name, string type, char cat);
    int getid() const;
    string getname() const;
    string gettype() const;
    virtual int getquantity();
    virtual int getdurability();
    virtual void add(int _quantity);
    virtual void subtract(int _quantity);
    virtual bool isFull();
    virtual bool isEmpty();
    bool isTool();
    bool isNonTool();
};

#endif

```

```

#include "Item.hpp"

#ifndef _NON_TOOL_HPP_
#define _NON_TOOL_HPP_

class NonTool : public Item {
private:
    int quantity;
    const int maxquantity;
public:
    NonTool();
    NonTool(int id, string name, string type, int quantity);
    int getquantity();
    int getmaxquantity();
    void add(int _quantity);
    void subtract(int _quantity);
    bool isFull();
    bool isEmpty();
};

#endif

```

```
#include "Item.hpp"

#ifndef _TOOL_HPP_
#define _TOOL_HPP_

class Tool : public Item {
private:
    int durability;
public:
    Tool();
    Tool(int id, string name, string type, int durability);
    int getdurability();
    bool isEmpty();
    void use();
    bool isDestroyed();
};

#endif
```

Konsep inheritance pada kasus ini digunakan untuk memberikan atribut quantity untuk NonTool dan atribut durability untuk Tool. Hal ini diperlukan karena atribut tersebut hanya dimiliki oleh NonTool (quantity) dan Tool (durability) saja.

Selain pada Item, Tool, dan NonTool, konsep inheritance juga digunakan pada Command dan semua *child class*-nya (Give, Discard, Move, Use, Craft, Export).

```
Command.hpp
1  #ifndef _COMMAND_HPP_
2  #define _COMMAND_HPP_
3
4  #include <iostream>
5  #include <fstream>
6  #include <string>
7  #include <regex>
8  #include <vector>
9  #include "Table.hpp"
10 #include "Libitem.hpp"
11 #include "LibRecipe.hpp"
12 #include "Exception/BaseException.hpp"
13 using namespace std;
14
15 class Command{
16 public:
17     virtual void printMessage() = 0;
18 };
19
20 class Give : public Command{
21 private:
22     int qty;
23     string name;
24 public:
25     Give(string name, int qty);
26     void Execute(Table<3,9> *inventory, libitem lib);
27     void printMessage();
28 };
29
30 class Discard : public Command{
31 private:
32     int qty;
33     string InvId;
34 public:
35     Discard(string InvId, int qty);
36     void Execute(Table<3,9> *inventory);
37     void printMessage();
38 };
39
40 class Move : public Command{
41 private:
42     string src;
43     int N;
44     vector<string> dest;
45 public:
46     Move(string src, int N, vector<string> dest);
47     void Execute(Table<3,9> *inventory, Table<3,3> *crafting);
48     void printMessage();
49 };
50
51 class Use : public Command{
52 private:
53     string InvID;
54 public:
55     Use(string InvID);
56     void Execute(Table<3,9> *inventory);
57     void printMessage();
58 };
```



```

60 class Craft : public Command{
61 public:
62     Craft();
63     void Execute(Table<3,3> *crafting, Table<3,9> *inventory, RecipeList rList, libitem lib);
64     void printMessage();
65 };
66
67 class Export : public Command{
68 private:
69     string fileName;
70     Table<3,9> inventory;
71 public:
72     Export(string fileName);
73     bool checkFilename();
74     void Execute(Table<3,9>* inventory);
75     void printMessage();
76 };
77
78 #endif
79

```

Alasan digunakan konsep inheritance pada Command dengan semua *child class*-nya adalah karena adanya kesamaan fungsi praktis dari semua command, yaitu untuk mengubah dan memodifikasi elemen-elemen utama di dalam program utama. Selain itu, penggunaan inheritance pada Command juga membantu modularitas dengan memungkinkan implementasi *pure virtual function* untuk message (akan dibahas lebih lanjut pada bagian *virtual functions*).

2.2. Method/Operator Overloading

```

NonTool.hpp
NonTool.hpp > ...
1  #include "Item.hpp"
2
3  #ifndef _NON_TOOL_HPP_
4  #define _NON_TOOL_HPP_
5
6  class NonTool : public Item {
7  private:
8      int quantity;
9      const int maxquantity;
10 public:
11     NonTool();
12     NonTool(int id, string name, string type, int quantity);
13     int getquantity();
14     int getmaxquantity();
15     void add(int _quantity);
16     void subtract(int _quantity);
17     bool isFull();
18     bool isEmpty();
19     void operator=(const NonTool other);
20 };
21
22 #endif

```

```

void NonTool::operator=(const NonTool other) {
    this->quantity = other.quantity;
    this->name = other.name;
    this->type = other.type;
    this->category = other.category;
}

```

Pada kelas NonTool, digunakan sebuah fungsi untuk *overload* operator “=”. Tujuannya adalah untuk menggantikan fungsi *copy constructor*. Alasan menggunakan *operator overloading* dan bukan *copy constructor* sejati adalah karena tidak ada atribut pada NonTool yang menggunakan memori secara rekursif, ditandai dengan penggunaan pointer.

Selain pada NonTool, operator overloading juga digunakan dalam kelas LibItem dan RecipeList. Berikut adalah bentuk header dan implementasinya:

```

8 // Include other header files
9 #include "Item.hpp"
10 #include "command.hpp"
11 #include "Libitem.hpp"
12 #include "Recipe.hpp"
13
14 #ifndef _LIBRECIPE_HPP_
15 #define _LIBRECIPE_HPP_
16
17 class RecipeList {
18     private:
19         vector<Recipe> List;
20     public:
21         // default constructor, isinya 0
22         RecipeList();
23         // Fungsi untuk mengisi RecipeList
24         void addRecipes(const Recipe newRecipe);
25         // Fungsi untuk mendapatkan resep yang cocok untuk crafting
26         bool checkPurePairTool(Table<3,3> *C);
27         Recipe checkCrafting(Table<3,3> *C);
28         void operator<<(const string& folder); //baca file
29         vector<filesystem::path> getFilesinFolder (string folder);
30         vector<string> splitString(string full);
31         void showAll();
32 };
33
34 #endif // _LIBRECIPE_HPP_

```

```

106 void RecipeList::operator<<(const string& folder){
107     string str;
108     vector<string> ingredients;
109     vector<filesystem::path> files = getFilesinFolder(folder);
110
111     for (int i=0;i<files.size();i++){
112         string filename{files[i].u8string()};
113         ifstream file(filename);
114         Recipe newRes;
115         if (file.is_open()){
116             getline(file,str,' ');
117             newRes.setRowSize(stoi(str));
118             getline(file,str);
119             newRes.setColSize(stoi(str));
120             for (int i=0;i<newRes.getRowSize();i++){
121                 getline(file,str);
122                 ingredients = splitString(str);
123                 for (int j=0;j<ingredients.size();j++){
124                     newRes.insertIngredient(i,j,ingredients[j]);
125                 }
126             }
127             getline(file,str,' ');
128             newRes.setRecipeResult(str);
129             getline(file,str);
130             newRes.setResultQty(stoi(str));
131         }
132         addRecipes(newRes);
133     }
134 }

```

```

#ifndef LIBITEM_HPP_
#define LIBITEM_HPP_

#include <iostream>
#include "Item.hpp"
#include "Tool.hpp"
#include "NonTool.hpp"
#include "Exception/BaseException.hpp"
#include <vector>
#include <iostream>
#include <fstream>
#include <string.h>
using namespace std;

using namespace std;

class libitem {
private:
    vector<Tool> tools;
    vector<NonTool> nontools;
public:
    libitem();
    void addItem(int ID, string name, string type, string category);
    NonTool searchnontoolsbyname(string name) const;
    NonTool searchnontoolsbyid(int id) const;
    Tool searchtoolsbyname(string name) const;
    Tool searchtoolsbyid(int id) const;
    void printlibitem();
    void operator<<(const string& fileName); //baca file
};

#endif

```

```

void libitem::operator<<(const string& fileName){
    int id;
    string name;
    string type;
    string category;
    ifstream file(fileName);
    if (file.is_open()){
        while(file >> id >> name >> type >> category){ //baca tiap line
            this->addItem(id, name, type, category);
        }
    }
}

```

Operator “<<” di-*overload* untuk menerima input string dari sebuah file yang berisikan Item untuk libitem, atau Recipe untuk LibRecipe. Penggunaan operator overloading digunakan karena sesuai dengan spesifikasi C++, dimana operator overloading hanya dapat menerima tepat 1 buah parameter di ruas kanan operator dan 1 buah parameter, yaitu *instance class* itu sendiri, di ruas kiri operator. Selain itu, penggunaan operator overloading juga memudahkan untuk membaca dan memahami isi dari *source code*.

2.3. Template & Generic Classes

Pada implementasi program ini, konsep *template* dan *generic class* digunakan pada *class Table*. Kelas tersebut diimplementasikan untuk membuat *instance Inventory* dan *Crafting Table*. Alasan digunakan fitur *generic class* pada komponen ini adalah karena secara praktis, tidak ada perbedaan jauh antara *Inventory* dan *Crafting Table*. Keduanya harus bisa menampung Item (baik itu berupa *Tool* maupun *NonTool*), harus bisa mengakses indeks pada *Table*, dan mengubah atribut dari *Item* pada indeks tertentu.

Perbedaan antara *Inventory* dengan *Crafting Table* hanya pada ukurannya. Ukuran dari *Inventory* adalah 3x9, dengan satu slot dapat menampung maksimum 1 buah *Tool* dan maksimum 64 buah *NonTool*. Sementara itu, ukuran dari *Crafting Table* adalah 3x3, dengan spesifikasi penyimpanan per slot sama dengan *Inventory*.

Salah satu keuntungan dari penggunaan *generic class* pada *Table* adalah memungkinkan *Inventory* dan *Crafting Table* menjadi dua objek yang berbeda meskipun didasari oleh *constructor* yang sama. Artinya, fungsi yang hanya menerima *Inventory* sebagai parameter tidak dapat menerima *CraftingTable* dan sebaliknya, sekalipun keduanya merupakan *instance* dari *Table*. Contoh pemanggilan fungsinya ada pada snippet kode berikut:

```

Table.hpp > Table<maxrow, maxcol> > useTool(int)
1  #include "Item.hpp"
2  #include "NonTool.hpp"
3  #include "Tool.hpp"
4  #include "Exception/BaseException.hpp"
5  #include <math.h>
6  #include <fstream>
7
8  #ifndef _TABLE_HPP_
9  #define _TABLE_HPP_
10
11  template <int maxrow, int maxcol>
12  class Table {
13      private:
14          Item** item;
15      public:
16          Table() {
17              this->item = new Item*[maxrow];
18              for (int i = 0; i < maxrow; i++) {
19                  this->item[i] = new Item*[maxcol];
20                  for (int j = 0; j < maxcol; j++) {
21                      this->item[i][j] = new NonTool();
22                  }
23              }
24          }
25
26          ~Table() {
27              delete[] item;
28          }
29

```

Konsep ini digunakan untuk membuat tabel berukuran apapun menjadi lebih mudah sehingga kelas tabel ini diimplementasikan pada pembuatan tabel inventory dan tabel crafting pada program.

2.4. Exception

Exception pada program ini menggunakan BaseException (terdapat dalam BaseException.hpp) sebagai parent dan memiliki anak kelas yang merupakan spesifikasi dari exception.

```

class BaseException {
protected:
    string expType;
public:
    BaseException();
    string getExpType() const;
    virtual void printMessage() = 0;
};

```

```
class InventoryFullException : public BaseException {
public:
    InventoryFullException();
    void printMessage() override;
};

class ItemNotFoundException : public BaseException {
private:
    int id;
    string name;
public:
    ItemNotFoundException(string name);
    ItemNotFoundException(int id);
    void printMessage() override;
};

class DiscardInvalidException : public BaseException {
private:
    int slotID;
public:
    DiscardInvalidException(int slotID);
    void printMessage() override;
};

class NotNonToolException : public BaseException {
private:
    Item* falseItem;
public:
    NotNonToolException(Item* failItem);
    void printMessage() override;
};
```

```

class NotToolException : public BaseException {
private:
    Item* falseItem;
public:
    NotToolException(Item* failItem);
    void printMessage() override;
};

class ItemInvalidException : public BaseException {
private:
    Item* srcItem;
    Item* destItem;
public:
    ItemInvalidException(Item* srcItem, Item* destItem);
    void printMessage() override;
};

class NotEmptySlotException : public BaseException {
private:
    int slotID;
    char type;
public:
    NotEmptySlotException(int slotID, char type);
    void printMessage() override;
};

```

```

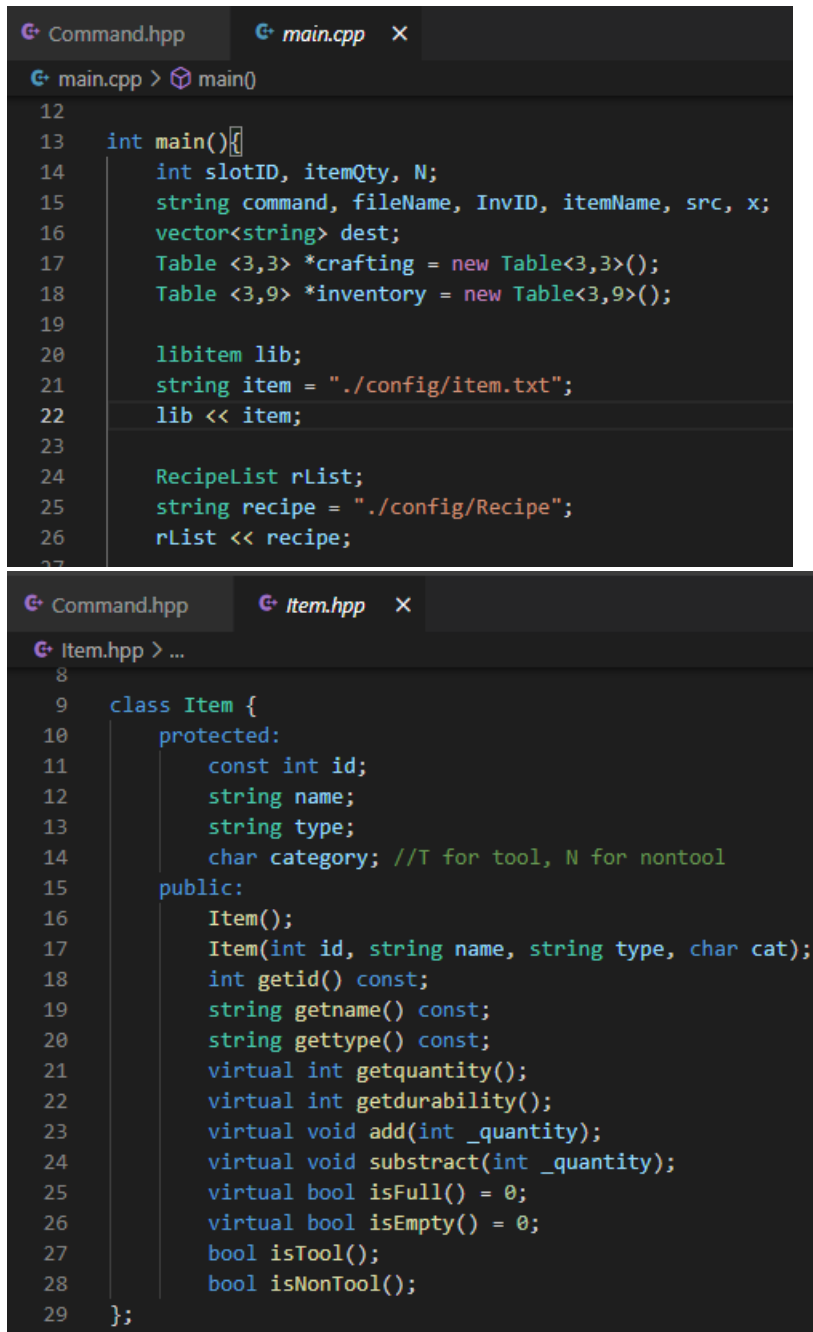
class EmptySlotException : public BaseException {
private:
    int slotID;
    char type;
public:
    EmptySlotException(int slotID, char type);
    void printMessage() override;
};

```

Exception digunakan sesuai dengan tipe exception yang diinginkan. Penamaan Exception merepresentasikan jenis exception yang diinginkan. Misalkan, ItemNotFoundException digunakan untuk memberikan exception ketika item tidak ditemukan dalam library item. Setiap exception melakukan override method printMessage dari kelas BaseException.

2.5. C++ Standard Template Library

Implementasi dari STL C++ pada program ini terpakai cukup dominan dalam bentuk string dan vector. Implementasi string dapat ditemukan pada hampir semua kelas, baik itu sebagai atribut dari kelas maupun sebagai parameter aliran I/O fungsi dari kelas. Beberapa contoh dari aplikasi string dapat dilihat pada snippet code di bawah:



```

Command.hpp  main.cpp x
main.cpp > main()
12
13 int main(){
14     int slotID, itemQty, N;
15     string command, fileName, InvID, itemName, src, x;
16     vector<string> dest;
17     Table<3,3> *crafting = new Table<3,3>();
18     Table<3,9> *inventory = new Table<3,9>();
19
20     libitem lib;
21     string item = "./config/item.txt";
22     lib << item;
23
24     RecipeList rList;
25     string recipe = "./config/Recipe";
26     rList << recipe;
27
Item.hpp x
Item.hpp > ...
8
9 class Item {
10     protected:
11         const int id;
12         string name;
13         string type;
14         char category; //T for tool, N for nontool
15     public:
16         Item();
17         Item(int id, string name, string type, char cat);
18         int getid() const;
19         string getname() const;
20         string gettype() const;
21         virtual int getquantity();
22         virtual int getdurability();
23         virtual void add(int _quantity);
24         virtual void subtract(int _quantity);
25         virtual bool isFull() = 0;
26         virtual bool isEmpty() = 0;
27         bool isTool();
28         bool isNonTool();
29 };

```

Alasan penggunaan library string cukup sederhana. Lebih sederhana bagi penulis dan pembaca kode ini untuk memahami *source code* apabila dituliskan dalam bentuk string dibandingkan jika dalam bentuk array of char. Selain itu, implementasi string juga memungkinkan penggunaan *built-in functions* dan *operator overloading* seperti untuk membandingkan dua string, atau mencari panjang sebuah string.

Berikut adalah beberapa contoh aplikasi STL vector pada program ini:

```
// Include other header files
#include "Item.hpp"
#include "command.hpp"
#include "Libitem.hpp"
#include "Recipe.hpp"

#ifndef _LIBRECIPE_HPP_
#define _LIBRECIPE_HPP_

class RecipeList {
private:
    vector<Recipe> List;
public:
    // default constructor, isinya 0
    RecipeList();
    // Fungsi untuk mengisi RecipeList
    void addRecipes(const Recipe newRecipe);
    // Fungsi untuk mendapatkan resep yang cocok untuk crafting
    bool checkPurePairTool(Table<3,3> *C);
    Recipe checkCrafting(Table<3,3> *C);
    void operator<<(const string& folder); //baca file
    vector<filesystem::path> getFilesinFolder (string folder);
    vector<string> splitString(string full);
    void showAll();
};

#endif // _LIBRECIPE_HPP_
```

```
command.hpp  Libitem.hpp X
libitem.hpp > libitem > nontools

#include <vector>
#include <iostream>
#include <fstream>
#include <string.h>
using namespace std;

using namespace std;

class libitem {
private:
    vector<Tool> tools;
    vector<NonTool> nontools;
public:
    libitem();
    void addItem(int ID, string name, string type, string category);
    NonTool searchnontoolsbyname(string name) const;
    NonTool searchnontoolsbyid(int id) const;
    Tool searchtoolsbyname(string name) const;
    Tool searchtoolsbyid(int id) const;
    void printlibitem();
    void operator<<(const string& fileName); //baca file
};

#endif
```

Penulis mengimplementasikan STL vector karena library vector pada C++ memungkinkan resizing. Ini membantu program penulis karena dengan ini tidak akan ada kebutuhan `max_size` jika menggunakan list statis. Selama Item yang digunakan terdaftar pada file input untuk libitem dan dapat dicraft sesuai Recipe pada LibRecipe serta adanya memori pada komputer pengguna, tidak ada batasan jumlah elemen vector yang dapat disimpan. Selain itu, penggunaan vector juga memudahkan penulis karena adanya *built-in functions* seperti `at()`, `front()`, `size()`, dll untuk membantu iterasi isi dari vector itu sendiri.

2.6. Abstract Base Class

Konsep ini digunakan pada kelas Item, Tool, dan NonTool. Kelas item adalah kelas yang menyimpan detail dari sebuah item yang terdapat pada game. Kelas NonTool merupakan kelas anak dari kelas Item yang menyimpan informasi quantity dan maxquantity yang menyatakan jumlah NonTool yang terdapat pada 1 objek. Kelas Tool merupakan kelas anak dari kelas Item yang menyimpan informasi durability yang menyatakan ketahanan dari sebuah Tool.

```

Item.hpp > Item > isEmpty()
1  #ifndef _ITEM_HPP_
2  #define _ITEM_HPP_
3
4  #include <iostream>
5  #include <string>
6
7  using namespace std;
8
9  class Item {
10     protected:
11         const int id;
12         string name;
13         string type;
14         char category; //T for tool, N for nontool
15     public:
16         Item();
17         Item(int id, string name, string type, char cat);
18         int getid() const;
19         string getname() const;
20         string gettype() const;
21         virtual int getquantity();
22         virtual int getdurability();
23         virtual void add(int _quantity);
24         virtual void subtract(int _quantity);
25         virtual bool isFull() = 0;
26         virtual bool isEmpty() = 0;
27         bool isTool();
28         bool isNonTool();
29     };
30
31 #endif

```

```

Tool.hpp > Tool > isFull()
1  #include "Item.hpp"
2
3  #ifndef _TOOL_HPP_
4  #define _TOOL_HPP_
5
6  class Tool : public Item {
7      private:
8          int durability;
9      public:
10         Tool();
11         Tool(int id, string name, string type, int durability);
12         int getdurability();
13         bool isFull();
14         bool isEmpty();
15         void use();
16         bool isDestroyed();
17     };
18
19     #endif

```

```

NonTool.hpp > ...
1  #include "Item.hpp"
2
3  #ifndef _NON_TOOL_HPP_
4  #define _NON_TOOL_HPP_
5
6  class NonTool : public Item {
7      private:
8          int quantity;
9          const int maxquantity;
10     public:
11         NonTool();
12         NonTool(int id, string name, string type, int quantity);
13         int getquantity();
14         int getmaxquantity();
15         void add(int _quantity);
16         void subtract(int _quantity);
17         bool isFull();
18         bool isEmpty();
19         void operator=(const NonTool other);
20     };
21
22     #endif

```

Pada kelas Item, diterapkan fungsi abstract yaitu isFull() dan isEmpty() yang akan diimplementasikan pada kelas Tool dan NonTool. Alasan menggunakan fungsi abstract adalah kondisi keadaan penuh pada Tool dan NonTool berbeda. Hal ini juga berlaku untuk kondisi keadaan kosong pada Tool dan NonTool.

2.7. Virtual (Polymorphism)

Polymorphism adalah salah satu fitur dari pemrograman berorientasi objek. Pada bagian ini, penulis berfokus pada implementasi dengan menggunakan *virtual functions*, baik itu yang bersifat *derived* atau yang bersifat *pure virtual*. *Polymorphism* memungkinkan fungsi yang bernama sama, berparameter sama, dan sama-sama dimiliki oleh *parent* dan *child* untuk menghasilkan atau melakukan proses yang berbeda. Contoh dari penggunaan *derived virtual* adalah pada snippet berikut:

```
Command.hpp X
Command.hpp > ...
15 class Command{
16 public:
17     virtual void printMessage() = 0;
18 };
19
20 class Give : public Command{
21 private:
22     int qty;
23     string name;
24 public:
25     Give(string name, int qty);
26     void Execute(Table <3,9> *inventory, libitem lib);
27     void printMessage();
28 };
```

Pada contoh diatas (untuk mempermudah penjelasan, hanya akan dibandingkan kelas *Command* dan *Give* sebagai *parent-child*), dilihat bahwa fungsi *printMessage()* merupakan sebuah fungsi yang *pure virtual*. Artinya, tidak ada implementasi fungsi dari *parent*, dan diserahkan sepenuhnya ke kelas-kelas yang meng-inherit fungsi tersebut. Ini memberikan keleluasaan bagi penulis untuk memodifikasi fungsi yang sudah ada untuk memenuhi spesifikasi kelas berbeda tanpa membuat fungsi baru. Berikut adalah implementasi *printMessage()* untuk kelas *Give* dan *Move* (juga merupakan *child class* dari *Command*) :

```
73 void Give::printMessage(){
74     cout << this->name << " berhasil diberikan" << endl;
75 }
76
```

```
void Move::printMessage(){
    char src = this->src[0];
    char dest = this->dest[0][0];

    if (src == 'I' && dest == 'I'){
        cout << "Barang berhasil ditumpuk" << endl;
    }
    else if(src == 'I' && dest == 'C'){
        cout << "Barang berhasil dipindahkan ke slot crafting" << endl;
    }
    else if(src == 'C' && dest == 'I'){
        cout << "Barang berhasil dipindahkan ke inventory" << endl;
    }
}
```

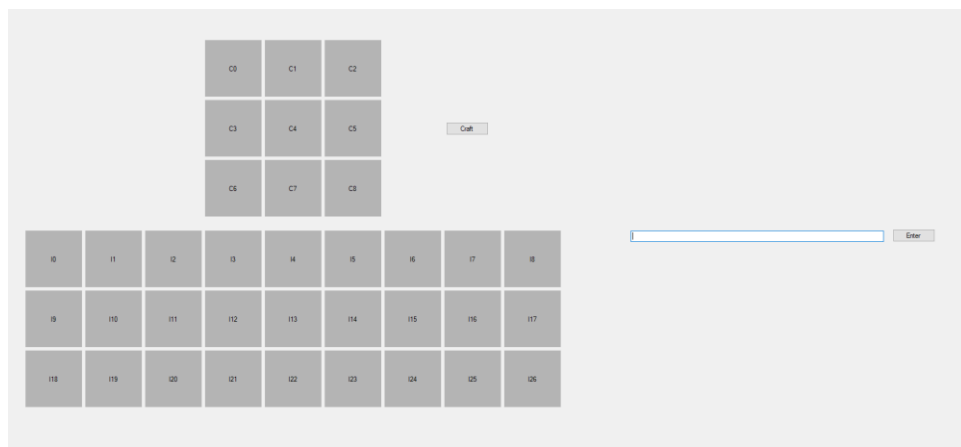
3. Bonus Yang dikerjakan

3.1. Bonus yang diusulkan oleh spek

3.1.1. Pembuatan GUI

GUI pada proram ini diperuntukan untuk memberikan visualisasi dari Inventory dan Crafting. Program tetap bisa dijalankan menggunakan CLI jika tidak ingin menggunakan GUI

GUI pada program ini menggunakan platform Windows Forms untuk bahasa C++. Windows Froms digunakan untuk memudahkan pembuatan dekstop application. Tampilan GUI pada program ini terlihat seperti berikut:



Command dapat dituliskan melalui text box di sebelah kanan dan menekan enter untuk mengeksekusi command. Command yang digunakan sama dengan command pada cli kecuali untuk Craft. Craft dapat dilakukan dengan menekan tombol Craft di samping kanan dari crafting Table.

3.1.2. Item dan Tool Baru

Untuk bonus ini, item dan tool baru yang ditambahkan ke dalam main program adalah gold ingot, gold nugget, golden sword, golden axe, golden pickaxe.

3.1.3. Unit Testing Implementation

Untuk bonus ini, masing-masing dari anggota tim penulis telah membuat dan mencoba secara mandiri proses debugging kelas yang masing-masing implementasikan. Dengan alasan kerapihan *source code*, file main tidak dilampirkan di repository yang akan disubmit. Namun, uji coba *unit testing* yang telah dilakukan membantu proses penulis menggabungkan semua kelas menjadi satu kesatuan.

4. Pembagian Tugas

Modul	Implementer	Tester
Item	13520004 13520064 13520133 13520136 13520160	13520004 13520064 13520133 13520136 13520160
Tool	13520004 13520064 13520133 13520136 13520160	
NonTool	13520004 13520064 13520133 13520136 13520160	
Table	13520064 13520160	
Libitem	13520004 13520064 13520133 13520136	
Recipe	13520004	
Librecipe	13520004 13520064 13520133	
Command	13520133	
Exception	13520136	
main	13520133 13520160	

Command	Implementer	Tester
SHOW	13520133 13520160	13520004 13520064 13520133 13520136 13520160
GIVE	13520133 13520136 13520160	

DISCARD	13520133 13520136 13520160	
MOVE	13520133 13520136 13520160	
USE	13520133 13520136	
CRAFT	13520133 13520136 13520160	
EXPORT	13520133 13520136 13520160	

Bonus	Implementer	Tester
GUI	13520136	13520136
ITEM & TOOL BARU	13520133	13520133
UNIT TESTING	13520004 13520064 13520133 13520136 13520160	13520004 13520064 13520133 13520136 13520160