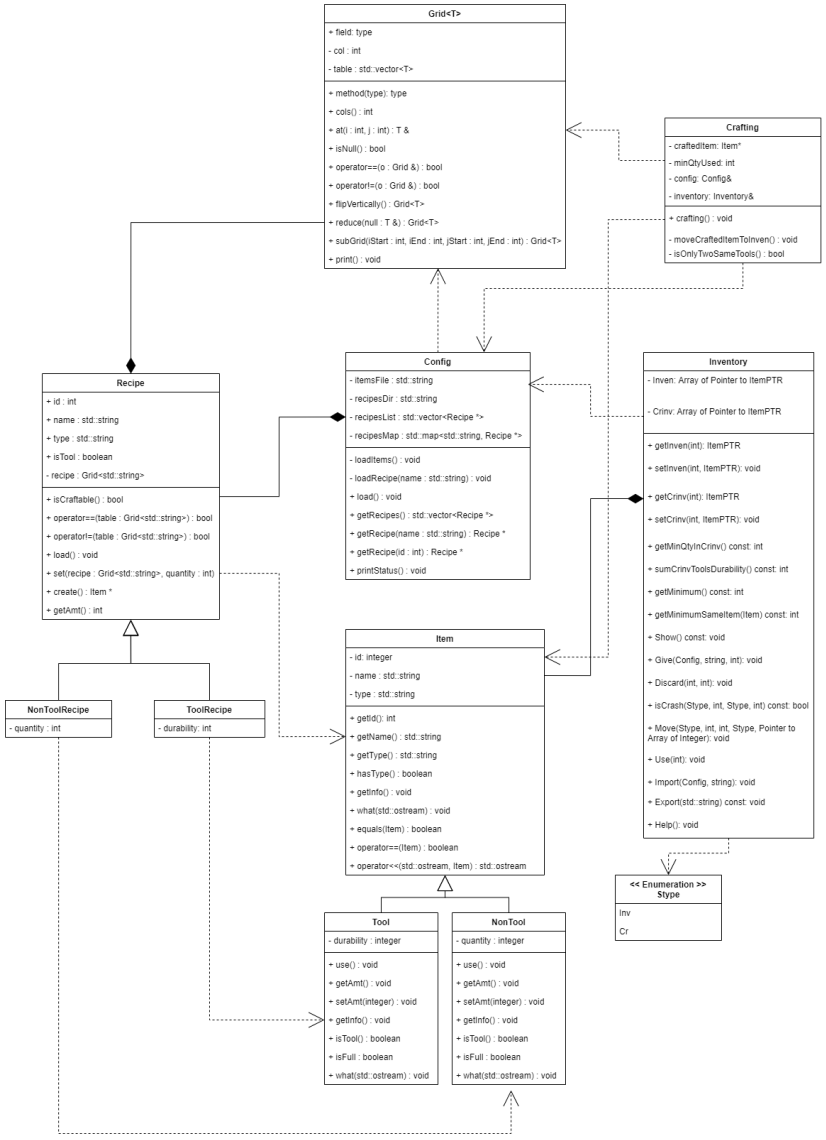
Kelas : 03
Nomor Kelompok : 11
Nama Kelompok : 13520129
1. 13520129 / Nathanael Santoso
2. 13520024 / Hilya Fadhilah Imania
3. 13520162 / Daffa Romyz Aufa
4. 13519210 / Dwi Kalam Amal Tauhid
5. 13517129 / Panawar Hasibuan
Asisten Pembimbing : Gregorius Jovan Kresnadi

1. Diagram Kelas

Gambarkan diagram kelas dari desain OOP lengkap dengan header method dan atribut kelas secara singkat. Anda dapat menggunakan *tools* apapun untuk menggambarkan diagram kelas. Pastikan diagram kelas **dapat terbaca**: tulisan tidak terlalu kecil, semua garis jelas, dan lainnya. Beberapa bahasa seperti C++ juga memiliki *tools* untuk me-*generate* diagram kelas secara otomatis, misalnya doxygen.

Jelaskan desain dari kelas Anda: alasan dipilihnya desain, kelebihan atau kekurangan (jika ada) diagram kelas yang dipilih, dan kendala yang dialami dalam pemilihan desain OOP.


Grid<T>
+ field: type
- col : int
- table : std::vector<T>
+ method(type): type
+ cols() : int
+ at(i : int, j : int) : T &
+ isNull() : bool
+ operator==(o : Grid &) : bool
+ operator!=(o : Grid &) : bool
+ flipVertically() : Grid<T>
+ reduce(null : T &) : Grid<T>
+ subGrid(iStart : int, iEnd : int, jStart : int, jEnd : int) : Grid<T>
+ print() : void
Crafting
- craftedItem: Item*
- minQtyUsed: int
- config: Config&
- inventory: Inventory&
+ crafting() : void
- moveCraftedItemToInven() : void
- isOnlyTwoSameTools() : bool
Recipe
+ id : int
+ name : std::string
+ type : std::string
+ isTool : boolean
- recipe : Grid<std::string>
+ isCraftable() : bool
+ operator==(table : Grid<std::string>) : bool
+ operator!=(table : Grid<std::string>) : bool
+ load() : void
+ set(recipe : Grid<std::string>, quantity : int)
+ create() : Item *
+ getAmt() : int
Config
- itemsFile : std::string
- recipesDir : std::string
- recipesList : std::vector<Recipe *>
- recipesMap : std::map<std::string, Recipe *>
- loadItems() : void
- loadRecipe(name : std::string) : void
+ load() : void
+ getRecipes() : std::vector<Recipe *>
+ getRecipe(name : std::string) : Recipe *
+ getRecipe(id : int) : Recipe *
+ printStatus() : void
Inventory
- Inven: Array of Pointer to ItemPTR
- Crinv: Array of Pointer to ItemPTR
+ getInven(int): ItemPTR
+ setInven(int, ItemPTR): void
+ getCrinv(int): ItemPTR
+ setCrinv(int, ItemPTR): void
+ getMinQtyInCrinv() const: int
+ sumCrinvToolsDurability() const: int
+ getMinimum() const: int
+ getMinimumSameItem(Item) const: int
+ Show() const: void
+ Give(Config, string, int): void
+ Discard(int, int): void
+ isCrash(Stype, int, Stype, int) const: bool
+ Move(Stype, int, int, Stype, Pointer to Array of Integer): void
+ Use(int): void
+ Import(Config, string): void
+ Export(std::string) const: void
+ Help(): void
Item
- id: integer
- name : std::string
- type : std::string
+ getId(): int
+ getName() : std::string
+ getType() : std::string
+ hasType() : boolean
+ getInfo() : void
+ what(std::ostream) : void
+ equals(Item) : boolean
+ operator==(Item) : boolean
+ operator<<(std::ostream, Item) : std::ostream
NonToolRecipe
- quantity : int
ToolRecipe
- durability: int
<< Enumeration >>
Stype
Inv
Cr
Tool
- durability : integer
+ use() : void
+ getAmt() : void
+ setAmt(integer) : void
+ getInfo() : void
+ isTool() : boolean
+ isFull : boolean
+ what(std::ostream) : void
NonTool
- quantity : integer
+ use() : void
+ getAmt() : void
+ setAmt(integer) : void
+ getInfo() : void
+ isTool() : boolean
+ isFull : boolean
+ what(std::ostream) : void

Gambar jelas dapat diakses melalui: https://drive.google.com/file/d/1pK4kJZrv4I4gRkd7D-aBo3eKKRG3K3G4/view?usp=sharing

Desain kelas dilakukan dengan melihat objek-objek utama apa saja yang perlu direpresentasikan. Berdasarkan persoalan Minecraft, maka objek dasar yang perlu dibuat adalah Item dan Inventory. Karena item ada dua jenis, tool dan nontool, yang keduanya memiliki perbedaan sifat dan kelakuan, maka dibuat menjadi kelas derivasi dari Item yaitu Tool dan NonTool. Selain itu karena program perlu untuk menyimpan daftar item dan resepnya, yang di-load dari file konfigurasi, maka dibuat kelas Config. Resep juga dibuat menjadi kelas sendiri Recipe untuk keperluan membuat item. Resep merupakan cetak biru berbentuk matriks yang perlu dimanipulasi sehingga dibuat kelas Grid sebagai representasi matriks. Selain itu karena item yang dihasilkan berbeda tergantung jenisnya, maka dibuat juga kelas derivasi ToolRecipe dan NonToolRecipe. Proses crafting yang cukup rumit juga dipertimbangkan sehingga dibuat kelas tersendiri yaitu Crafting.

Kelebihan dari implementasi ini adalah abstraksi yang cukup jelas pada beberapa kelas, terutama Recipe. Dengan desain pada kelas Recipe, kode program yang mengimplementasikan crafting tidak perlu lagi repot-repot melakukan identifikasi manual apakah item-item pada crafting table sama dengan suatu resep item tertentu. Recipe juga dapat menghasilkan item sesuai dengan tipenya.

Kekurangan dari implementasi ini adalah beberapa redundansi dan inkonsistensi yang terjadi, terutama dalam penanganan Item. Kode program yang menangani Item masih harus melakukan pengecekan terhadap tipe Item, apakah Tool atau NonTool. Oleh karena itu berarti abstraksi yang diberikan oleh kelas Item masih kurang, apalagi mempertimbangkan Item adalah kelas abstrak.

2. Penerapan Konsep OOP

2.1. Inheritance & Polymorphism

2.1.1. Exception

```
class CrashSlotException : public Exception
```

Exception memiliki abstract base class Exception yang memudahkan catch jika ada Exception yang dithrow. Exception juga memiliki atribut id untuk mengidentifikasi nomor Exceptionnya dan juga pure virtual function what() yang diimplementasikan child class untuk mengembalikan string berisi detil error.

2.1.2. Item

```
class NonTool : public Item
```

```
class Tool : public Item
```

Item memiliki dua kategori yaitu tool dan nontool. Item tool dan item nontool akan memiliki atribut yang sama dan beberapa atribut class sendiri. Oleh karena itu, konsep inheritance cocok untuk diimplementasikan pada kasus ini. Parent class nya adalah Item. Child class adalah Tool dan NonTool.

2.1.3. Recipe

```
class ToolRecipe : public Recipe
```

```
class NonToolRecipe : public Recipe
```

Recipe sama seperti Item, memiliki dua kategori yaitu tool dan nontool. Hal ini agar saat diminta, resep sebagai factory bisa menghasilkan item baru sesuai dengan tipenya. Parent: Recipe (abstract). Child: ToolRecipe dan NonToolRecipe.

2.2. Method/Operator Overloading

2.2.1. Grid

Grid meng-overload operator equality yaitu “==” dan “!=” untuk membandingkan value dari dua buah Grid, apakah sama atau tidak. Perbandingan dilakukan seperti perbandingan matriks biasa. Namun ada satu hal kecil yang menarik. Pada overloading operator ==, Grid tidak meminta objek Grid yang dibandingkan mengandung tipe elemen yang sama. Objek yang diminta adalah Grid<U>, bukan Grid<T>.

```
template <class U>
bool operator==(const Grid<U> &o) const
{
  if (row ≠ o.rows() || col ≠ o.cols())
    return false;

  for (int i = 0; i < row; ++i)
  {
    for (int j = 0; j < col; ++j)
    {
      if (at(i, j) ≠ o.at(i, j))
        return false;
    }
  }

  return true;
}
```

Oleh karena itu, dua tipe berbeda Grid<T1> dan Grid<T2> dapat dibandingkan value-nya dengan benar selama terdapat operator equality yang dapat membandingkan value T1 dan T2.

2.2.2. Inventory

Inventory mengoverload getMinimum karena ada dua kemungkinan, mengambil indeks minimum kosong atau mengambil index minimum nontool yang sama.

```
const int Inventory::getMinimum() {
    for (int i = 0; i < 27; i++) {
        if (this->Inven[i] == nullptr) return i;
    }
    return -1;
} // Gets index of item of minimum empty slot

const int Inventory::getMinimum(Item& I) {
    for (int i = 0; i < 27; i++) {
        if (this->Inven[i] != nullptr && *this->Inven[i] == I && !this->Inven[i]->isFull()) return i;
    }
    return -1;
} // Gets index of item of minimum same item
```

2.2.3. Item

```
bool operator==(const Item &);
friend bool operator==(const Item *, const std::string&);
friend bool operator!=(const Item *, const std::string&);
```

Berguna untuk membandingkan dua jenis barang agar implementasi Move dan crafting lebih mudah.

```
friend std::ostream &operator<<(std::ostream &os, const Item *item);
```

Berguna agar dapat memakai sintaks yang lebih simpel untuk mengeluarkan data dari item yang memudahkan implementasi Show() pada inventory

2.2.4. Recipe

Recipe meng-overload operator equality, sama seperti Grid. Recipe memanfaatkan method-method Grid untuk membandingkan Grid yang diminta dan Grid resep item, yaitu reduce() dan flipVertically(). Hal ini memudahkan kasus pada resep yang bersifat submatrix, sehingga tidak perlu ditangani atau dipusingkan oleh pemanggil. Kedua Grid juga dibandingkan dengan operator equality yang telah dijelaskan sebelumnya.

```
bool Recipe::operator==(const Grid<Item *> &table) const
{
  if (!isCraftable())
    return false;

  auto reduced = table.reduce(nullptr);
  return reduced == recipe ||  reduced == recipe.flipVertically();
}
```

2.3. Template & Generic Classes

2.3.1. Grid

```
bool Recipe::operator==(const Grid<Item *> &table) const
{
    if (!isCraftable())
        return false;

    auto reduced = table.reduce(nullptr);
    return reduced == recipe || reduced == recipe.flipVertically();
}
```

Grid merupakan tipe data dengan konsep matriks, yang memiliki baris dan kolom. Karena grid adalah suatu tipe dasar layaknya list/vector, map, dan set, maka cocok untuk diimplementasikan sebagai generic class. Konsep-konsep matriks yang diaplikasikan terhadap grid juga tidak terpengaruh oleh tipe datanya.

2.4. Exception

2.4.1. Abstract Base Class

```
class Exception
{
protected:
    const int id;
public:
    Exception(int id) : id(id) {}
    ~Exception() {}

    int getId() const { return this->id; }
    virtual std::string what() const = 0;
};
```

Abstract Base Class untuk Exception, dipakai sebagai template untuk exception lain dan membuat error handling lebih mudah karena hanya perlu catch base class. Secara umum exception diterapkan agar program tidak berhenti namun command diabaikan.

2.4.2. CrashSlotException

```
class CrashSlotException : public Exception
{
public:
    CrashSlotException() : Exception(1) {}
    ~CrashSlotException() {}

    std::string what() const { return "Something different already in destination slot."; }
};
```

Exception ini dithrow ketika user melakukan command move namun slot yang dituju memiliki tool atau nontool jenis sama namun penuh atau jenis item nontool berbeda.

2.4.3. NumberTooBigException

```
class NumberTooBigException : public Exception
{
public:
    NumberTooBigException() : Exception(2) {}
    ~NumberTooBigException() {}

    std::string what() const { return "Number inputed was too big."; }
};
```

Exception ini dithrow ketika jumlah masukan lebih dari jumlah yang ada misalnya jika user melakukan attempt untuk discard item lebih banyak daripada yang ada maka akan mengeluarkan exception ini. Exception ini juga dipakai di move khususnya dari inventory ke crafting.

2.4.4. NotExistsException

```
class NotExistsException : public Exception
{
public:
    NotExistsException() : Exception(3) {}
    ~NotExistsException() {}

    std::string what() const { return "Input does not exist."; }
};
```

Exception ini dithrow ketika slot asal kosong dan digunakan oleh Move, Discard, Craft, dan pembacaan command. Contoh penggunaannya adalah jika user melakukan discard di slot I0 namun slot tersebut kosong atau jika melakukan craft tapi crafting table kosong.

2.4.5. WrongTypeException

```
class WrongTypeException : public Exception
{
public:
    WrongTypeException() : Exception(4) {}
    ~WrongTypeException() {}

    std::string what() const { return "Input of the wrong type."; }
};
```

Digunakan ketika input salah tipe dan dipakai oleh use, jika user melakukan use menggunakan nontool, maka exception ini yang dilempar.

2.4.6. ContainerFullException

```
class ContainerFullException : public Exception
{
public:
    ContainerFullException() : Exception(5) {}
    ~ContainerFullException() {}

    std::string what() const { return "Object Buffer is already full type."; }
};
```

Digunakan ketika inventory penuh namun masih ada item yang perlu masuk ke inventory. Digunakan dalam command Give dan Craft agar inventory konsisten.

2.4.7. InvalidQuantityException

```
class InvalidQuantityException : public Exception
{
public:
    InvalidQuantityException() : Exception(6) {}
    ~InvalidQuantityException() {}

    std::string what() const { return "Invalid quantity."; }
};
```

Exception ini dithrow ketika kuantitas masukan invalid seperti jika melakukan GIVE namun hanya memberikan nama item dan tidak memberikan kuantitas. Exception ini dipakai dalam command input exception handling

2.4.8. ConfigFileException

```
class ConfigFileException : public Exception
{
public:
    ConfigFileException() : Exception(7) {}
    ~ConfigFileException() {}

    std::string what() const { return "Error accessing config file."; }
};
```

Exception ini dithrow ketika Class Config tidak bisa melakukan loading resep dan item sehingga akan error. Exception ini diimplementasikan agar dapat mengetahui error pada config dan program tidak tutup tanpa ada error message apapun.

2.4.9. CraftedItemIsNotFound

```
class CraftedItemNotFound : public Exception
{
public:
    CraftedItemNotFound() : Exception(8) {}
    ~CraftedItemNotFound() {}

    std::string what() const { return "No item is crafted yet."; }
};
```

Exception ini berkemungkinan ter-*invoke* ketika **crafting** *method* dieksekusi, yaitu jika tidak terdapat item yang dibentuk. Selain *exception* ini, terdapat satu situasi lainnya yang memungkinkan *throwing exception*, **NotExistsException**, yakni ketika *crafting inventory* kosong. Walaupun berdasarkan pesan *error* yang ditampilkan **NotExistsException** dapat mengakomodirnya, namun untuk membedakan pemberian informasi *error* jika program mengalami salah satu situasi tersebut maka *exception* ini menjadi solusi.

2.4.10. InvalidCommand

```
class InvalidCommandException : public Exception
{
public:
    InvalidCommandException() : Exception(9) {}
    ~InvalidCommandException() {}

    std::string what() const { return "Invalid command."; }
};
```

Exception ini dithrow ketika ada command apapun yang tidak dituliskan dalam menu help atau help sendiri. Misalkan user melakukan input “HELLO” akan melempar exception ini.

2.4.11. IgnoredArgement

```
class IgnoredArgumentsException : public Exception
{
public:
    IgnoredArgumentsException() : Exception(10) {}
    ~IgnoredArgumentsException() {}

    std::string what() const { return "Excessive arguments are ignored."; }
};
```

Exception ini dithrow ketika masukan pengguna kebanyakan dengan contoh “GIVE WOODEN_SWORD 1 1”, maka satu terakhir akan diabaikan namun tetap diberi tahu bahwa ada kesalahan tersebut.

2.5. C++ Standard Template Library

2.5.1. Config

Pada kelas Config digunakan dua struktur, yaitu vector dan map, untuk menyimpan daftar resep. Keduanya bertipe pointer to Recipe. Jadi yang disimpan di memory hanya satu buah objek untuk tiap Recipe, namun ditunjuk oleh vector dan map. Map berguna untuk pencarian Recipe berdasarkan nama karena sering digunakan. Sedangkan vector dibutuhkan untuk kebutuhan iterasi yang terurut.

2.5.2. Grid

Grid, meskipun berupa matriks dengan baris dan kolom, disimpan secara internal menggunakan vector. Rumus yang digunakan sederhana yaitu elemen (i,j) berada pada posisi (i x jumlah kolom + j) dalam vector. Ini memudahkan implementasi karena vector mudah dimanipulasi dan tidak perlu menangani construct, copy, assignment, dan destruct.

2.6. Konsep OOP lain

2.6.1. Abstract Base Class

Abstract base class diimplementasikan pada kelas Item dan Recipe, yang keduanya memiliki variasi Tool dan Nontool. Ini berguna karena resep yang disimpan dalam daftar resep serta item yang disimpan dalam inventory tidak mempedulikan tipenya.

2.6.2. Composition

Composition digunakan dalam banyak kelas. Contohnya Grid merupakan komponen dari Recipe, dan Recipe sendiri merupakan komponen dari Config. Hal ini dapat dilihat pada diagram kelas. Pemecahan masalah ini berguna karena tiap kelas bisa fokus pada satu fungsi utamanya. Misal Config berfungsi load dan menyimpan data resep. Recipe sendiri berfungsi untuk keperluan crafting. Sedangkan Grid berfungsi sebagai matriks yang dapat dimanipulasi.

3. Bonus Yang dikerjakan

3.1. Bonus yang diusulkan oleh spek

3.1.1. Multiple Crafting

Secara garis besar, pendekatan yang dilakukan adalah dengan mengecek minimum kuantitas dari seluruh item di *crafting inventory* (dengan status awal *crafting inventory* tidak kosong dan konfigurasi item di *crafting inventory* sesuai dengan salah satu resep *crafting*). Hal tersebut menjadi penentu berapakah kuantitas item yang akan dihasilkan dengan mengalikan minimum kuantitas tersebut dengan *default* jumlah item yang terbentuk. Jika item telah terbentuk maka kuantitas item-item di *crafting inventory* dikurangi dengan minimum kuantitas tersebut. Jika

kuantitasnya 0 untuk suatu item maka item tersebut dihapus dari crafting inventory. Sementara jika lebih dari 0, item tersebut akan memiliki kuantitas setelah dilakukan pengurangan tadi.

Gambar berikut adalah kode untuk menentukan minimum kuantitas dari item-item di *crafting inventory.*

```
if (*recipe == crinvConfig){
    this->minQtyUsed = this->inventory.getMinQtyInCrinv();
```

Gambar berikut adalah metode yang dipanggil dalam menentukan minimum kuantitas dari item-item di *crafting inventory.*

```
int Inventory::getMinQtyInCrinv() const{
    int min = 0;
    for (int i = 0; i < 9; ++i){
        if (Crinv[i] && Crinv[i]->isTool()){
            return 1;
        }
        else if (Crinv[i] && !Crinv[i]->isTool()){
            if (min == 0){
                min = Crinv[i]->getAmt();
            }
            else if (min != 0 && Crinv[i]->getAmt() < min){
                min = Crinv[i]->getAmt();
            }
        }
    }
    return min;
}
```

Gambar berikut adalah *instance* kode untuk menentukan berapakah kuantitas *crafted item* yang dihasilkan.

```
itemCraftedQty = recipe->getAmt() * minQtyUsed;
this->craftedItem = new NonTool(recipe->id, recipe->name, recipe->type, itemCraftedQty);
```

Gambar berikut adalah *instance* kode untuk mengurangi kuantitas atau menghapus item-item di *crafting inventory*.

```
// Remove some or all items in Crinven
for (int i = 0; i < 9; ++i){
    if (this->inventory.getCrinv(i)){
        if (this->inventory.getCrinv(i)->isTool()){
            this->inventory.DeleteSlotContents(this->inventory.Cr, i);
        } else {
            int remainingQty = this->inventory.getCrinv(i)->getAmt() - this->minQtyUsed;
            if (remainingQty == 0){
                this->inventory.DeleteSlotContents(this->inventory.Cr, i);
            } else {
                this->inventory.getCrinv(i)->setAmt(remainingQty);
            }
        }
    }
}
```

3.1.2. Unit Testing Implementation

Unit testing dilakukan dengan membuat driver setiap modul yang mengecek fungsi fungsinya. Untuk memudahkan digunakan tool makefile dalam kompilasi. Karena tiap modul unit test memerlukan kode modul yang berbeda-beda, include file “.cpp” atau source code dilakukan langsung di dalamnya. Sebagai contoh, ini adalah penggalan awal dari unit test untuk modul Recipe

```cpp
#include <iostream>
#include <string>
#include "../lib/Recipe.hpp"

#include "../lib/Recipe.cpp"
#include "../lib/Item.cpp"
#include "../lib/NonTool.cpp"
#include "../lib/Tool.cpp"

using namespace mobicraft;

int main(void)
{
  std::cout << "\nCreate non-tool recipe.\n";
  Recipe *nt = new NonToolRecipe(1, "OAK_PLANK", "PLANK");

  std::cout << "\nCreate recipe grid.\n";
  Grid<std::string> ntGrid(1, 1);
  ntGrid.at(0, 0) = "OAK_LOG";

  nt->set(ntGrid, 4);
  ntGrid.print();
```

Untuk mengkompilasi dan menjalankan unit test, misal untuk modul resep yang terdapat pada “tests/recipe.cpp”, cukup melakukan perintah

```
$ make unit-test MODULE=recipe
```

3.2 Bonus Kreasi Mandiri

3.2.1 Import function

```
void Inventory::Import(Config& c, std::string path) {
    std::ifstream inf(path);
    std::string line, token, num;
    std::string delimiter = ":";

    for (int i = 0; i < 9; i++) {
        if (this->Crinv[i] != nullptr) this->DeleteSlotContents(Cr, i);
    }

    for (int i = 0; i < 27; i++) {
        if (this->Inven[i] != nullptr) this->DeleteSlotContents(Inv, i);
    }

    while(std::getline(inf, line)) {
        token = line.substr(0, line.find(delimiter));
        num = line.erase(0, line.find(delimiter) + delimiter.length());
        if (num != "0") {
            const Recipe *r = c.getRecipe(std::stoi(token));
            if (r->isTool) {
                this->Inven[this->getMinimum()] = new Tool(r->id, r->name, r->type, std::stoi(num));
            }
            else this->Give(c, r->name, std::stoi(num));
        }
    }

    std::cout << std::endl << "Import Successful" << std::endl;

} // Import Inventory dari file .txt
```

Untuk menemani command Export, maka dibuat juga command import yang menerima file keluaran export dan memasukkan barang ke dalam inventory sesuai format Export. Terlebih dahulu isi inventory dan crafting akan dihapus kemudian barang dimasukkan ke dalam Inventory. Disarankan penggunaan jika ingin mengimplementasikan sistem save.

3.2.2 Export Terlebih Dahulu Memindahkan Item dari Crafting Slot ke Inventory

```
try {
    for (int i = 0; i < 9; i++) {
        if (this->Crinv[i] != nullptr) {
            idx[0] = this->getMinimum(*this->Crinv[i]);
            if (idx[0] == -1) idx[0] = this->getMinimum();
            this->Move(Cr, i, 1, Inv, idx);
        }
    }
} catch (Exception *e) {
    std::cout << std::endl << e->what() << std::endl;
}
```

Agar lebih konsisten, maka item yang masih di dalam crafting slot ketika Export dipanggil akan berpindah ke dalam inventory agar item-item tersebut tidak terbuang. Namun jika mendapatkan **ContainerFullException**, item sisa yang belum dimasukkan ke dalam inventory akan terbuang layaknya game asli ketika item terlempar di tanah

4. Pembagian Tugas

Modul (dalam poin spek)	Implementer	Tester
Item	13520162	13529162
Recipe	13520024	13520024
Config	13520024	13520024
Inventory	13520129	13520129
Crafting	13519210, 13517129	13519210
Unit Testing	13520024, 13520129	13520024, 13520129
Test & Check	13520129	13520129