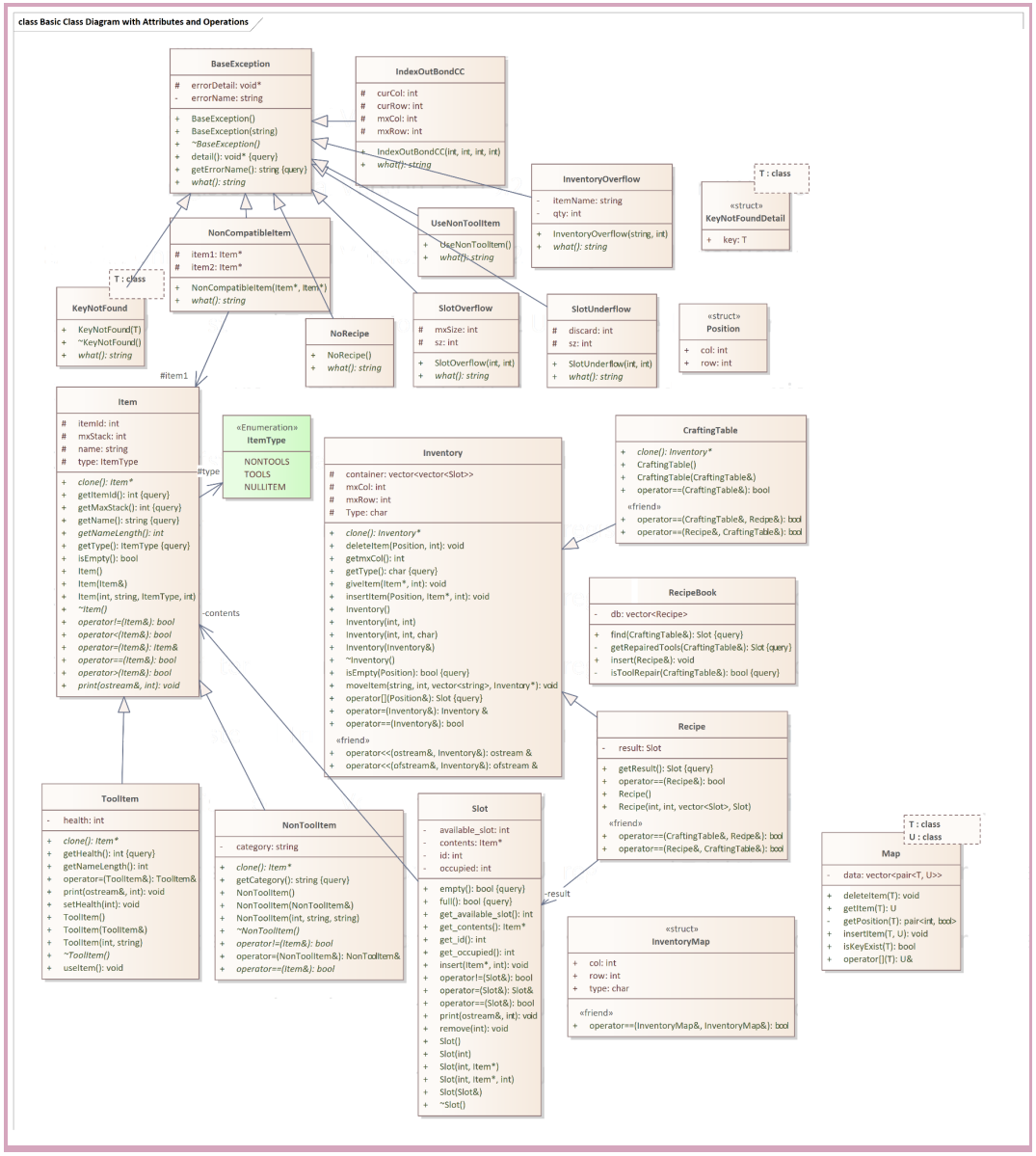
Kelas : 02
Nomor Kelompok : 10
Nama Kelompok : Tambang Jaya
1. 13520047 / Hana Fathiyah
2. 13520053 / Yohana Golkaria Nainggolan
3. 13520095 / Firizky Ardiansyah
4. 13520098 / Andika Naufal Hilmy
5. 13520128 / Bayu Samudra
Asisten Pembimbing: Muhammad Fauzan Rafi Sidiq Widjonarto
Tautan *Repository* Github: https://github.com/bayusamudra5502/tubes-oop
Tautan Notulensi Asistensi:
https://docs.google.com/document/d/1yiKgWQS_xrgm1pCKuCL_mpOyplbGzhb-/edit?usp=sharing&ouid=107993402028950311532&rtpof=true&sd=true

1. Diagram Kelas

Tautan Tampilan Diagram Kelas:
https://github.com/bayusamudra5502/tubes-oop/blob/734574ed9047b7976f6b77d94cc10530e12fb154/doc/Class%20Diagram/Class%20Diagram.md


class Basic Class Diagram with Attributes and Operations
BaseException
errorDetail: void*
- errorName: string
+ BaseException()
+ BaseException(string)
+ ~BaseException()
+ detail(): void* {query}
+ getErrorName(): string {query}
+ what(): string
IndexOutBondCC
curCol: int
curRow: int
mxCol: int
mxRow: int
+ IndexOutBondCC(int, int, int, int)
+ what(): string
InventoryOverflow
- itemName: string
- qty: int
+ InventoryOverflow(string, int)
+ what(): string
T : class
«struct»
KeyNotFoundDetail
+ key: T
UseNonToolItem
+ UseNonToolItem()
+ what(): string
NonCompatibleItem
item1: Item*
item2: Item*
+ NonCompatibleItem(Item*, Item*)
+ what(): string
T : class
KeyNotFound
+ KeyNotFound(T)
+ ~KeyNotFound()
+ what(): string
NoRecipe
+ NoRecipe()
+ what(): string
SlotOverflow
mxSize: int
sz: int
+ SlotOverflow(int, int)
+ what(): string
SlotUnderflow
discard: int
sz: int
+ SlotUnderflow(int, int)
+ what(): string
«struct»
Position
+ col: int
+ row: int
#item1
Item
itemId: int
mxStack: int
name: string
type: ItemType
+ clone(): Item*
+ getItemId(): int {query}
+ getMaxStack(): int {query}
+ getName(): string {query}
+ getNameLength(): int
+ getType(): ItemType {query}
+ isEmpty(): bool
+ Item()
+ Item(Item&)
+ Item(int, string, ItemType, int)
+ ~Item()
+ operator!=(Item&): bool
+ operator<(Item&): bool
+ operator=(Item&): Item&
+ operator==(Item&): bool
+ operator>(Item&): bool
+ print(ostream&, int): void
«Enumeration»
ItemType
NONTOOLS
TOOLS
NULLITEM
#type
Inventory
container: vector<vector<Slot>>
mxCol: int
mxRow: int
Type: char
+ clone(): Inventory*
+ deleteItem(Position, int): void
+ getmxCol(): int
+ getType(): char {query}
+ giveItem(Item*, int): void
+ insertItem(Position, Item*, int): void
+ Inventory()
+ Inventory(int, int)
+ Inventory(int, int, char)
+ Inventory(Inventory&)
+ ~Inventory()
+ isEmpty(Position): bool {query}
+ moveItem(string, int, vector<string>, Inventory*): void
+ operator[](Position&): Slot {query}
+ operator=(Inventory&): Inventory &
+ operator==(Inventory&): bool
«friend»
+ operator<<(ostream&, Inventory&): ostream &
+ operator<<(ofstream&, Inventory&): ofstream &
CraftingTable
+ clone(): Inventory*
+ CraftingTable()
+ CraftingTable(CraftingTable&)
+ operator==(CraftingTable&): bool
«friend»
+ operator==(CraftingTable&, Recipe&): bool
+ operator==(Recipe&, CraftingTable&): bool
-contents
RecipeBook
- db: vector<Recipe>
+ find(CraftingTable&): Slot {query}
- getRepairedTools(CraftingTable&): Slot {query}
+ insert(Recipe&): void
- isToolRepair(CraftingTable&): bool {query}
Recipe
- result: Slot
+ getResult(): Slot {query}
+ operator==(Recipe&): bool
+ Recipe()
+ Recipe(int, int, vector<Slot>, Slot)
«friend»
+ operator==(CraftingTable&, Recipe&): bool
+ operator==(Recipe&, CraftingTable&): bool
ToolItem
- health: int
+ clone(): Item*
+ getHealth(): int {query}
+ getNameLength(): int
+ operator=(ToolItem&): ToolItem&
+ print(ostream&, int): void
+ setHealth(int): void
+ ToolItem()
+ ToolItem(ToolItem&)
+ ToolItem(int, string)
+ ~ToolItem()
+ useItem(): void
NonToolItem
- category: string
+ clone(): Item*
+ getCategory(): string {query}
+ NonToolItem()
+ NonToolItem(NonToolItem&)
+ NonToolItem(int, string, string)
+ ~NonToolItem()
+ operator!=(Item&): bool
+ operator=(NonToolItem&): NonToolItem&
+ operator==(Item&): bool
Slot
- available_slot: int
- contents: Item*
- id: int
- occupied: int
+ empty(): bool {query}
+ full(): bool {query}
+ get_available_slot(): int
+ get_contents(): Item*
+ get_id(): int
+ get_occupied(): int
+ insert(Item*, int): void
+ operator!=(Slot&): bool
+ operator=(Slot&): Slot&
+ operator==(Slot&): bool
+ print(ostream&, int): void
+ remove(int): void
+ Slot()
+ Slot(int)
+ Slot(int, Item*)
+ Slot(int, Item*, int)
+ Slot(Slot&)
+ ~Slot()
-result
«struct»
InventoryMap
+ col: int
+ row: int
+ type: char
«friend»
+ operator==(InventoryMap&, InventoryMap&): bool
T : class
U : class
Map
- data: vector<pair<T, U>>
+ deleteItem(T): void
+ getItem(T): U
- getPosition(T): pair<int, bool>
+ insertItem(T, U): void
+ isKeyExist(T): bool
+ operator[](T): U&

Deskripsi kelas:

Nama Kelas	Deskripsi
Item	Kelas yang menunjukkan satu jenis barang dan merupakan basis bagi ToolItem dan NonToolItem
ToolItem	Kelas turunan Item yang merupakan barang *tool*
NonToolItem	Kelas turunan Item yang merupakan barang *non-tool*
Slot	Kelas yang berfungsi sebagai tempat penyimpanan untuk satu item
Inventory	Kelas yang berfungsi sebagai tempat penyimpanan item yang dimiliki pemain
CraftingTable	Kelas yang berfungsi sebagai tempat pembuatan/*crafting* item
Recipe	Kelas yang berfungsi sebagai resep pembuatan satu jenis item
RecipeBook	Kelas yang berfungsi sebagai koleksi resep pembuatan item
Map	Kelas yang berfungsi sebagai pemetaan objek terhadap objek lainnya
BaseException	Kelas dasar yang membentuk berbagai jenis *exception*
NonCompatibleItem	Kelas *exception* yang muncul ketika item tidak kompatibel
KeyNotFound	Kelas *exception* yang muncul ketika *key* tidak ditemukan
NoRecipe	Kelas *exception* yang muncul ketika resep/*recipe* tidak ada
SlotOverflow	Kelas *exception* yang muncul ketika tumpukan item yang ada di slot melebihi kapasitas
SlotUnderflow	Kelas *exception* yang muncul ketika tumpukan item yang ada di slot kurang dari nol
UseNonToolItem	Kelas *exception* yang muncul ketika sebuah NonToolItem digunakan

IndexOutBondCC	Kelas *exception* yang muncul ketika indeks berada di luar batas/*range*
InventoryOverflow	Kelas *exception* yang muncul ketika inventory sudah tidak mampu lagi menampung item

Setiap kelas merepresentasikan entitas yang ada berupa barang, resep, tempat penyimpanan, dan berbagai *exception*. Pemilihan representasi entitas sebagai kelas bermaksud untuk memudahkan programmer dalam memahami interaksi antar-entitas pada objek-objek yang dibuat. Selain itu, tiap kode dalam kelas dibuat sesimpel mungkin dan dengan jumlah baris seminimal mungkin agar satu file tidak terlalu panjang dan rumit untuk dibaca. Karena pemilihan sistem tersebut, programmer akan lebih mudah memahami perilaku objek dan memperbaiki perilaku objek apabila ada kesalahan dalam pemrograman. Namun, programmer harus membuka lebih dari satu file karena kebanyakan kelas akan mereferensi pada kelas lainnya yang terdapat dalam file yang berbeda.

2. Penerapan Konsep OOP

2.1. Inheritance & Polymorphism

Kami menerapkan konsep inheritance dan polymorphism pada seluruh kasus exception, kelas CraftingTable, NonToolItem, dan ToolItem. Seluruh kasus *exception* dibuat untuk memastikan program dapat tetap berjalan meskipun *runtime* terganggu. Algoritma CraftingTable merupakan algoritma yang dibuat untuk menyediakan tempat untuk melakukan *crafting*. CraftingTable meng-*inherit* clone() method dari Inventory. Adapun cuplikan program dari kelas CraftingTable adalah sebagai berikut:

```
CraftingTable::CraftingTable() : Inventory(3, 3, 'C') {}

CraftingTable::CraftingTable(const CraftingTable& ct) : Inventory(ct) {}
Inventory* CraftingTable::clone() { return new CraftingTable(*this); }
```

NonToolItem merupakan kelas yang menyimpan algoritma untuk menyimpan fungsi-fungsi yang akan digunakan oleh *non-tool item*, seperti clone() method, getCategory() yang berfungsi untuk mengembalikan kategori item serta operator overloading yang akan dijelaskan di

bagian selanjutnya. NonToolItem meng-*inherit* clone() method, operator overloading =, ==, dan != dari kelas Item. Cuplikan program dari kelas NonToolItem adalah sebagai berikut:

```
Item* NonToolItem::clone() { return new NonToolItem(*this); }

string NonToolItem::getCategory() const { return this->category; }

NonToolItem& NonToolItem::operator=(const NonToolItem& other) {
  Item::operator=(other);
  this->category = other.category;
  return *this;
}

bool NonToolItem::operator==(const Item& other) {
  if (other.getType() != this->getType()) {
    return false;
  } else if (other.getItemId() == CATEGORY_ID ||
             this->getItemId() == CATEGORY_ID) {
    const NonToolItem& o = dynamic_cast<const NonToolItem&>(other);
    string thisCategory = this->getCategory();
    return o.getCategory() == thisCategory;
  } else if (Item::operator==(other)) {
    return true;
  }

  return false;
}

bool NonToolItem::operator!=(const Item& other) { return !((*this) == other); }
```

NonToolItem merupakan kelas yang menyimpan algoritma untuk menyimpan fungsi-fungsi yang akan digunakan oleh *tool item*, seperti fungsi untuk menggunakan *tool item*. ToolItem meng-*inherit* clone() method, getNameLength() method, fungsi useItem(), dan fungsi print() dari kelas Item. Cuplikan program dari kelas ToolItem adalah sebagai berikut:

```
Item* ToolItem::clone() { return new ToolItem(*this); }

void ToolItem::useItem() {
  if (this->health > 0) {
    this->health--;
  }
}

int ToolItem::getHealth() const { return this->health; }

void ToolItem::setHealth(int newHealth) { this->health = newHealth; }

int ToolItem::getNameLength() {
  int cnt = 0;
  if (this->getHealth() < 10) {
    cnt = 1;
  } else {
    cnt = 2;
  }
  cnt += this->getName().size() + 2;
  return cnt;
}

void ToolItem::print(ostream& s, int mxLength) {
  s << this->name << "(" << this->getHealth() << ")";
  for (int i = this->getNameLength(); i < mxLength; i++) {
    s << " ";
  }
}
```

Keuntungan penggunaan konsep *inheritance* dalam program ini adalah tidak harus menyalin semua data dan method dari suatu kelas jika akan menggunakannya lagi dan jika hendak melakukan modifikasi program, hanya perlu dilakukan di kelas induknya saja. Sedangkan keuntungan penggunaan *polymorphism* adalah untuk mencegah suatu method agar tidak dapat di-*override* oleh kelas-kelas turunannya.

2.2. Method/Operator Overloading

Kami mengimplementasikan method/operator overloading pada algoritma Inventory, Item, NonToolItem, Recipe, dan RecipeEquality. Kelas Inventory dibuat untuk menyimpan berbagai slot, membantu memasukan item pada posisi tertentu, tetapi jika penuh, membantu untuk memasukannya di slot selanjutnya yg masih kosong, melakukan deleteItem tanpa menghilangkan indeksnya. Kelas Inventory menggunakan method overloading ==, []. Cuplikan program dari kelas Inventory adalah sebagai berikut:

```
bool Inventory::operator==(Inventory &cc) {
  bool output = this->Type == cc.Type;
  if (this->mxCol == cc.mxCol && this->mxRow == cc.mxRow) {
    int i = 0;
    while (i < this->mxRow && output) {
      int j = 0;
      while (j < this->mxCol && output) {
        if (this->container[i][j].operator!=(cc.container[i][j])) {
          output = false;
        } else {
          j++;
        }
      }
      i++;
    }
  } else {
    output = false;
  }
  return output;
}

Slot Inventory::operator[](const Position &pos) const {
  if (pos.row > this->mxRow || pos.row < 0 || pos.col > this->mxCol ||
      pos.col < 0) {
    throw new IndexOutBondCC(mxRow, mxCol, pos.row, pos.col);
  }
  return this->container[pos.row][pos.col];
}
```

Kelas Item dibuat untuk menyimpan nama item dan jenis item yang memiliki kelas turunan Toolitem dan NonToolItem. Kelas ini juga bisa mengecek apakah sebuah item merupakan NullItem atau tidak, menyimpan nilai max stack dari item pada sebuah slot, dan menyimpan id item. Kelas Item menggunakan operator overloading ==, <, >, !=, = yang memiliki fungsi *assignment* dan melakukan pengecekan dua item. Adapun cuplikan program dari kelas Item adalah sebagai berikut:

```
bool Item::operator==(const Item& other) {
  return (this->itemId == other.itemId);
}

bool Item::operator<(const Item& other) {
  return (this->itemId < other.itemId);
}

bool Item::operator>(const Item& other) {
  return (this->itemId > other.itemId);
}

bool Item::operator!=(const Item& other) {
  return (this->itemId != other.itemId);
}

Item& Item::operator=(const Item& other) {
  this->mxStack = other.mxStack;
  this->itemId = other.itemId;
  this->name = other.name;
  this->type = other.type;
  return *this;
}
```

Kelas NonToolItem merupakan kelas turunan dari kelas Item. Kelas ini menyimpan semua fungsi dan method yang akan digunakan oleh item bertipe NonToolItem, seperti clone() method, fungsi untuk mendapatkan kategori dari item yang digunakan. Method operator overloading yang digunakan di kelas NonToolItem adalah operator overloading =, ==, dan != yang memiliki fungsi *assignment* dan pengecekan dua jenis item memiliki tipe yang sama atau tidak maupun kategori yang sama maupun tidak.

```
NonToolItem& NonToolItem::operator=(const NonToolItem& other) {
  Item::operator=(other);
  this->category = other.category;
  return *this;
}

bool NonToolItem::operator==(const Item& other) {
  if (other.getType() != this->getType()) {
    return false;
  } else if (other.getItemId() == CATEGORY_ID ||
             this->getItemId() == CATEGORY_ID) {
    const NonToolItem& o = dynamic_cast<const NonToolItem&>(other);
    string thisCategory = this->getCategory();
    return o.getCategory() == thisCategory;
  } else if (Item::operator==(other)) {
    return true;
  }

  return false;
}

bool NonToolItem::operator!=(const Item& other) { return !((*this) == other); }
```

Kelas Recipe merupakan kelas yang menyimpan algoritma yang berfungsi untuk menyimpan resep *crafting* dan jumlahnya. Kelas Recipe memiliki operator overloading =, == untuk melakukan *assignment* dan pengecekan apakah resep yang dimasukkan sudah benar. Untuk cuplikan program dari kelas Recipe dapat dilihat di bawah ini:

```
bool Recipe::operator==(const Recipe& r) {
  if (r.mxCol != this->mxCol || r.mxRow != this->mxCol ||
      this->result != r.result) {
    return false;
  }

  for (int i = 0; i < r.mxRow; i++) {
    for (int j = 0; j < r.mxCol; j++) {
      Slot s = r.container[i][j];
      if (s != this->container[i][j]) {
        return false;
      }
    }
  }

  return true;
}
```

Kelas RecipeEquality merupakan kelas yang khusus untuk melakukan pengecekan kecocokan resep yang digunakan dengan *crafting* yang dilakukan. Kelas ini hanya memiliki satu method, yaitu metode overloading == yang fungsinya sama dengan fungsi kelas ini. Cuplikan program untuk kelas ini dapat dilihat di bawah ini:

```
bool operator==(const CraftingTable& ct, const Recipe& r) {
  for (int i = 0; i + r.mxRow <= ct.mxRow; i++) {
    for (int j = 0; j + r.mxCol <= ct.mxCol; j++) {
      CraftingTable ctNew, ctNewReverse;

      for (int k = 0; k < r.mxRow; k++) {
        for (int l = 0; l < r.mxCol; l++) {
          Position pos = {k + i, l + j};
          Slot content = r.container[k][l];
          ctNewReverse.insertItem(pos, content.get_contents(),
                                  content.get_occupied());
        }
      }

      ctNew = ctNewReverse;
      for (int k = 0; k < ct.mxRow; k++) {
        reverse(ctNewReverse.container[k].begin(),
                ctNewReverse.container[k].end());
      }

      if (ctNew == ct || ctNewReverse == ct) {
        return true;
      }
    }
  }

  return false;
}

bool operator==(const Recipe& r, const CraftingTable& ct) { return ct == r; }
```

Keuntungan penggunaan operator overloading dalam pembuatan program ini adalah fleksibilitas dari operator overloading itu sendiri. Operator overloading memberikan kemungkinan untuk memberikan argumen secara fleksibel tergantung keadaan dan sesuai kebutuhan program.

2.3. Template & Generic Classes

Kami mengimplementasikan konsep template & kelas generik pada algoritma Map. Map yang kami rancang merupakan sebuah struktur data *dictionary* yang memungkinkan insersi, delesi, dan akses elemen dengan rata-rata kompleksitas O(log n). Meskipun C++ menyediakan pustaka yang bisa menggantikan struktur data ini, kami tetap mengimplementasikannya secara mandiri agar algoritma lebih fleksibel dan

mudah dikontrol. Pengimplementasian kelas Map ini juga dilakukan dalam file header. Tujuan dari aksi ini adalah mencegah perlunya mendefinisikan tipe data setiap kali akan menggunakan kelas Map ini. Adapun cuplikan program dari kelas Map, sebagai berikut.

```hpp
#pragma once

#include <exception/KeyNotFound.hpp>
#include <utility>
#include <vector>

using namespace std;

template <class T, class U>
class Map {
 private:
  vector<pair<T, U>> data;
  pair<int, bool> getPosition(T key);

 public:
  U& operator[](T key);
  U getItem(T key);
  void insertItem(T key, U value);
  void deleteItem(T key);
  bool isKeyExist(T key);
};

template <class T, class U>
U& Map<T, U>::operator[](T key) {
  pair<int, bool> pos = this->getPosition(key);

  if (pos.second) {
    return (this->data[pos.first].second);
  } else {
    throw new KeyNotFound(key);
  }
}

template <class T, class U>
U Map<T, U>::getItem(T key) {
  return (*this)[key];
}
```

```hpp
template <class T, class U>
void Map<T, U>::deleteItem(T key) {
  pair<int, bool> pos = this->getPosition(key);

  if (pos.second) {
    this->data.erase(this->data.begin() + pos.first);
  } else {
    throw new KeyNotFound(key);
  }
}

template <class T, class U>
void Map<T, U>::insertItem(T key, U value) {
  this->data.push_back(make_pair(key, value));
  int i = this->data.size() - 1;

  while (i > 0 && this->data[i] < this->data[i - 1]) {
    pair<T, U> tmp = this->data[i];
    this->data[i] = this->data[i - 1];
    this->data[i - 1] = tmp;
    i--;
  }
}

template <class T, class U>
bool Map<T, U>::isKeyExist(T key) {
  return this->getPosition(key).second;
}
```

```hpp
template <class T, class U>
pair<int, bool> Map<T, U>::getPosition(T key) {
  int p = 0, q = this->data.size();
  if(q==0){
    return make_pair(-1, false);
  }
  while (p < q && this->data[(p + q) / 2].first != key) {
    T keyData = this->data[(p + q) / 2].first;
    if (keyData > key) {
      q = (p + q) / 2;
    } else {
      p = (p + q) / 2 + 1;
    }
  }

  if (this->data[(p + q) / 2].first == key) {
    return make_pair((p + q) / 2, true);
  }

  return make_pair(-1, false);
}
```

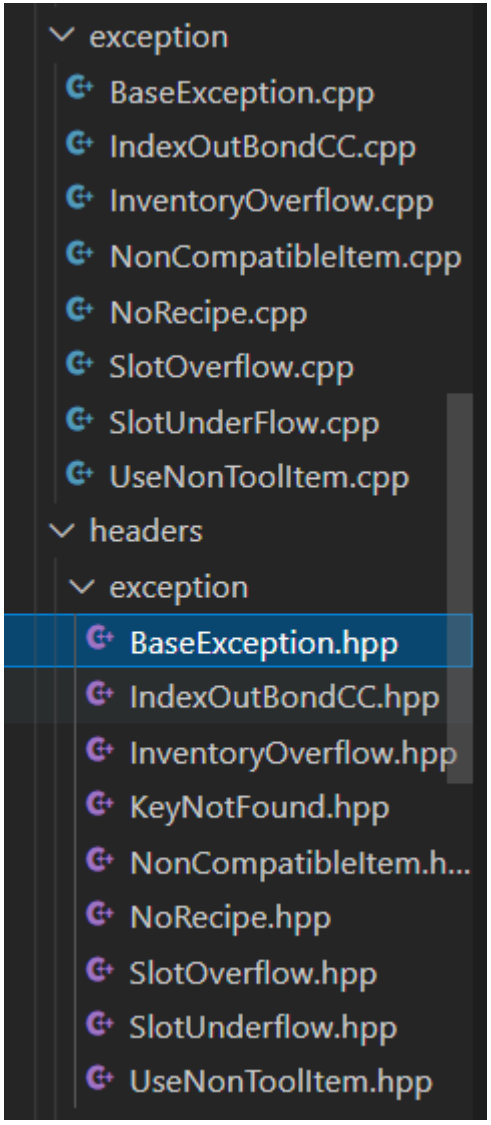
Keuntungan menggunakan konsep ini dalam implementasi Map adalah fleksibilitas tipe data atribut yang digunakan dalam *dictionary*. Perlu diperhatikan, aktualisasi beberapa *method* pada kelas ini memerlukan operator komparasi seperti <, > atau ==, sehingga tipe data atribut perlu memiliki overloading terhadap operator tersebut. Keuntungan lainnya adalah dengan adanya *method* getPosition, memungkinkan pencarian indeks dari elemen dengan kompleksitas O(log n), lebih cepat dibandingkan pencarian manual.

2.4. Exception

Kami mengimplementasikan exception menggunakan konsep *abstract class* dan *polymorphism*. Exception dibangun berdasarkan cetak biru basis kelas abstrak yang dijadikan sebagai *interface*. Jenis-jenis exception diciptakan berdasarkan kelas basis tersebut dan diimplementasikan sesuai dengan cetak biru kelas basis. Kelas exception juga bisa menambahkan *method* atau atribut baru untuk keperluan *error message passing*. Setiap kondisi yang memungkinkan adanya exception, program akan melakukan *throw* pointer terhadap kelas exception yang bersesuaian dan exception di-*catch* dengan konsep *polymorphism.* Adapun cetak biru kelas basis adalah sebagai berikut.

```cpp
#pragma once
#include <string>
using namespace std;

class BaseException {
 private:
  string errorName;

 protected:
  void* errorDetail;

 public:
  BaseException();
  BaseException(string errorName);
  virtual ~BaseException();
  string getErrorName() const;
  const void* detail() const;

  virtual string what() = 0;
};
```

Adapun jenis-jenis exception yang kami ciptakan adalah sebagai berikut.


exception
BaseException.cpp
IndexOutBondCC.cpp
InventoryOverflow.cpp
NonCompatibleItem.cpp
NoRecipe.cpp
SlotOverflow.cpp
SlotUnderFlow.cpp
UseNonToolItem.cpp
headers
exception
BaseException.hpp
IndexOutBondCC.hpp
InventoryOverflow.hpp
KeyNotFound.hpp
NonCompatibleItem.h...
NoRecipe.hpp
SlotOverflow.hpp
SlotUnderflow.hpp
UseNonToolItem.hpp

Di bawah ini adalah cuplikan cetak biru dan aktualisasi salah satu jenis exception yang diimplementasikan.

```hpp
#pragma once

#include <exception/BaseException.hpp>

class IndexOutBondCC: public BaseException {
    protected:
        int mxRow;
        int mxCol;
        int curRow;
        int curCol;

    public:
        IndexOutBondCC(int mxRow, int mxCol, int curRow, int curCol);
        virtual string what();
};
```

```cpp
#include <exception/IndexOutBondCC.hpp>
#include <sstream>

IndexOutBondCC::IndexOutBondCC(int mxRow, int mxCol, int curRow, int curCol) : BaseException("Index out of bond in inventory") {
    this -> mxRow = mxRow;
    this -> mxCol = mxCol;
    this -> curRow = curRow;
    this -> curCol = curCol;
}

string IndexOutBondCC::what() {
    stringstream ss;
    ss << this->getErrorName() << " has occured! the maximum capacity is (" << this->mxRow << ", " << this->mxCol << "), but you inserted (" << this->curRow << ", " << this->curCol << ").";
    return ss.str();
}
```

Dengan menggunakan *virtual method* what(), exception yang kami implementasikan memungkinkan untuk melakukan catch error cukup dengan tipe data pointer dari base class dan message yang ditampilkan ke layar adalah message sesuai jenis exception dari objek yang ditunjuk pointer base class. Berikut adalah contoh throw, try, dan catch exception dari program kami.

```cpp
Slot Inventory::operator[](const Position &pos) const {
    if (pos.row > this->mxRow || pos.row < 0 || pos.col > this->mxCol ||
        pos.col < 0) {
        throw new IndexOutBondCC(mxRow, mxCol, pos.row, pos.col);
    }
    return this->container[pos.row][pos.col];
}
```

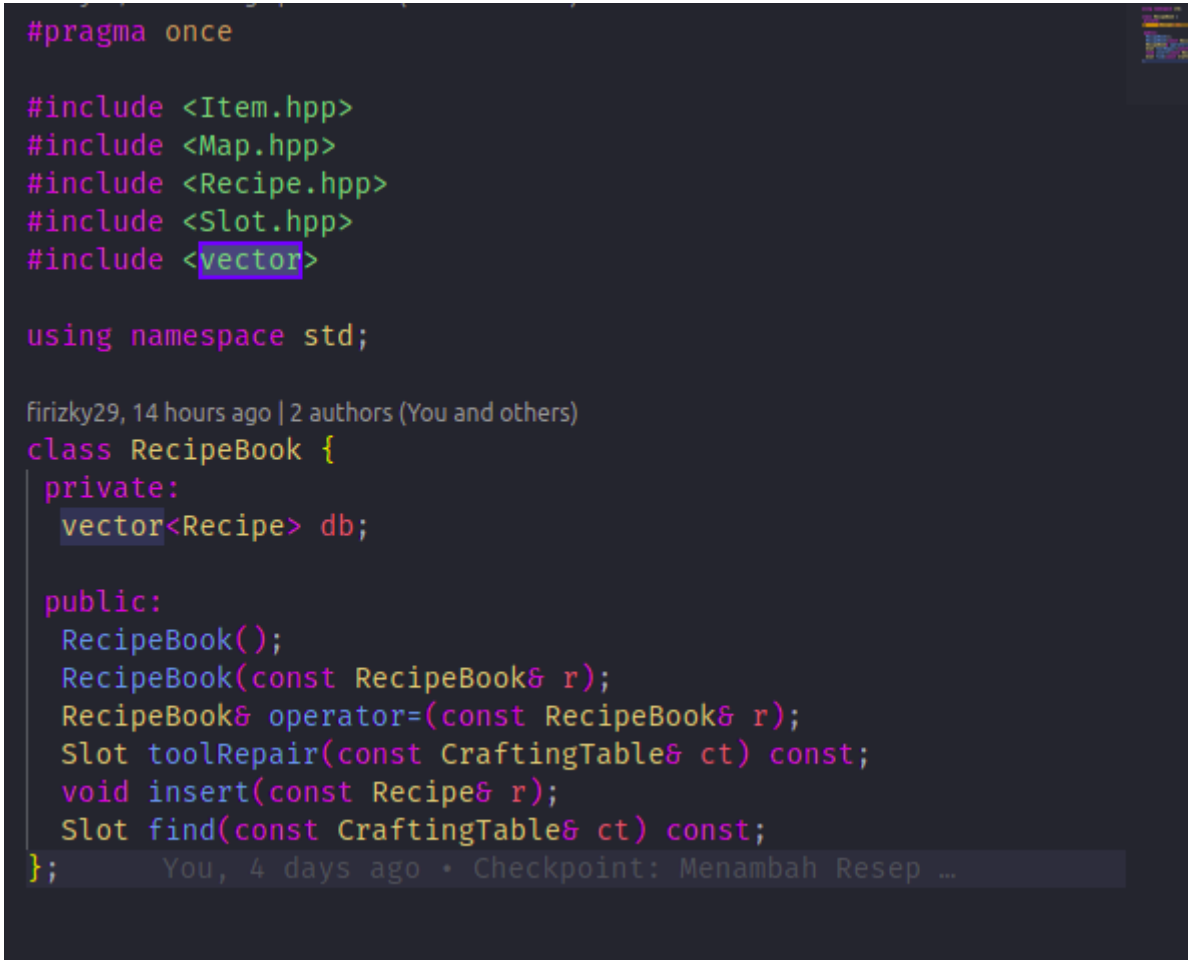
```
18  void doMultiCraft(CraftingTable& crafting, Inventory& inventory, RecipeBook recipes){
19    int i = 0;
20    while(true){
21      try {
22        doCraft(crafting, inventory, recipes);
23      } catch (BaseException* e) {
24        if(i==0){
25          throw e;
26        }
27        else{
28          break;
29        }
30      }
31      i++;
32    }
33  }
```

```
      } catch (BaseException* e) {
        cout << e->what() << "\n";
      }
    }
```

Keuntungan menggunakan exception pada program adalah memungkinkan tetap berjalannya program meskipun *runtime* terganggu, proses *abort* suatu algoritma juga dapat ditangani. Pesan yang diberikan oleh exception juga memudahkan proses *debugging* dengan mengetahui alasan dari kesalahan yang terjadi, sehingga tindak lanjutnya lebih mudah dan bisa segera dilakukan.

2.5. C++ Standard Template Library : Bayu

Kami mengimplementasikan C++ STL pada berbagai kelas. Beberapa STL yang kami gunakan adalah `vector<T>` dan `pair<T,U>`. Penggunaan STL vector pada umumnya digunakan untuk mengganti struktur data array dengan array dinamis sehingga dapat mencegah kesalahan dalam mengimplementasikan struktur data array ini. Penggunaan pair ditujukan untuk menyimpan data yang memiliki relasi satu sama lain. Berikut ini adalah contoh antarmuka kelas yang mengimplementasikan vector pada program.



```cpp
#pragma once

#include <Item.hpp>
#include <Map.hpp>
#include <Recipe.hpp>
#include <Slot.hpp>
#include <vector>

using namespace std;

firizky29, 14 hours ago | 2 authors (You and others)
class RecipeBook {
 private:
  vector<Recipe> db;

 public:
  RecipeBook();
  RecipeBook(const RecipeBook& r);
  RecipeBook& operator=(const RecipeBook& r);
  Slot toolRepair(const CraftingTable& ct) const;
  void insert(const Recipe& r);
  Slot find(const CraftingTable& ct) const;
};      You, 4 days ago • Checkpoint: Menambah Resep …
```

Pada kelas ini, penyimpanan data resep disimpan menggunakan array dinamis vector. Penambahan elemen dapat dilakukan pada method insert dengan menggunakan method vector `.push_back()`. Proses pencarian elemen pun dapat mengakses menggunakan operator kurung siku untuk mengambil resep dari vector.

Penggunaan STL pair digunakan pada kelas Map. Berikut ini adalah antarmuka kelas Map.

```cpp
#pragma once

#include <exception/KeyNotFound.hpp>
#include <utility>
#include <vector>
    You, 4 days ago • Komat kamit …
using namespace std;

You, 4 days ago | 1 author (You)
template <class T, class U>
class Map {
 private:
  vector<pair<T, U>> data;
  pair<int, bool> getPosition(T key);

 public:
  U& operator[](T key);
  U getItem(T key);
  void insertItem(T key, U value);
  void deleteItem(T key);
  bool isKeyExist(T key);
};
```

Pair pada kelas ini digunakan sebagai hubungan relasi antara key dan value pada elemen map.

2.6. Konsep OOP lain

Pada program ini, kami menerapkan konsep Abstract Base Class pada kelas BaseException. Hal ini dikarenakan kelas ini memiliki sebuah fungsi abstrak yaitu what yang nilainya dapat dikustomisasi sesuai dengan keperluan pada setiap kelas anaknya. Berikut adalah antarmuka dari kelas yang dimaksud.

```
#pragma once
#include <string>
using namespace std;

...
class BaseException {
 private:
  string errorName;

 protected:
  void* errorDetail;

 public:
  BaseException();
  BaseException(string errorName);
  virtual ~BaseException();
  string getErrorName() const;
  const void* detail() const;

  virtual string what() = 0;
};
```

Dalam program ini juga, kami melakukan Agregasi pada kelas RecipeBook dan Recipe. Recipe merupakan bagian dari RecipeBook. Akan tetapi, Recipe merupakan instance yang terpisah dari RecipeBook sehingga memiliki siklus hidup yang terpisah. Berikut ini adalah antarmuka dari kelas ini.

```cpp
#pragma once

#include <Item.hpp>
#include <Map.hpp>
#include <Recipe.hpp>
#include <Slot.hpp>
#include <vector>

using namespace std;

firizky29, 14 hours ago | 2 authors (You and others)
class RecipeBook {
 private:
  vector<Recipe> db;

 public:
  RecipeBook();
  RecipeBook(const RecipeBook& r);
  RecipeBook& operator=(const RecipeBook& r);
  Slot toolRepair(const CraftingTable& ct) const;
  void insert(const Recipe& r);
  Slot find(const CraftingTable& ct) const;
};      You, 4 days ago • Checkpoint: Menambah Resep …
```

Setiap resep baru akan dibuat diluar dari kelas ini. Kelas ini hanya bertanggung jawab untuk menyimpan kumpulan resep dan mencari resep saat akan dilakukannya crafting.

Program ini juga melakukan konsep Komposisi, yaitu pada kelas Slot. Kelas Slot menggunakan kelas Item untuk menyimpan informasi barang yang ia simpan. Kelas Item ini bisa dibentuk pada saat instansiasi kelas maupun diluar instansiasi kelas Slot. Pada saat

objek dari kelas slot ini akan dihapus, kelas Item ini juga akan ikut dihapus demi menjaga memori pada program. Berikut ini adalah antarmuka dari kelas ini

```
class Slot {
 private:
  int id;
  Item* contents;      You, 4 days ago • Checkpoint:
  int available_slot;
  int occupied;

 public:
  Slot();
  Slot(int id);
  Slot(int id, Item* item);
  Slot(int id, Item* item, int used);
  Slot(const Slot& other);
  ~Slot();
  Slot& operator=(Slot& other);
  bool operator==(const Slot& other);
  bool operator≠(const Slot& other);

  bool empty() const;
  bool full() const;
  void insert(Item* item, int count);
  void remove(int count);

  int get_id();
  Item* get_contents();
  int get_available_slot();
  int get_occupied();
  void useItem();

  void print(ostream& stream, int mxLen);

  friend void swap(Slot& a, Slot& b);
};
```

3. Bonus Yang dikerjakan

3.1. Bonus yang diusulkan oleh spek

3.1.1. Multiple Crafting

Untuk mengimplementasikan bonus ini, kami membuat sebuah modul yang melakukan crafting secara berlapis hingga sisa resource yang tersedia di dalam crafting table tidak dapat lagi diklaim hasilnya. Pendekatan yang dilakukan adalah dengan melakukan *loop* untuk melakukan crafting berkali-kali hingga resource crafting tidak menghasilkan apa-apa. Hasil crafting diakumulasikan dan disimpan dalam inventory. Berikut adalah cuplikan modul multiple crafting

```
18  void doMultiCraft(CraftingTable& crafting, Inventory& inventory, RecipeBook recipes){
19    int i = 0;
20    while(true){
21      try {
22        doCraft(crafting, inventory, recipes);
23      } catch (BaseException* e) {
24        if(i==0){
25          throw e;
26        }
27        else{
28          break;
29        }
30      }
31      i++;
32    }
33  }
```

Exception dilakukan untuk memberikan pesan error jika pada iterasi pertama resource tidak menghasilkan apa-apa.

3.1.2. Item dan Tool Baru

Dalam program dikenalkan beberapa Item dan Tools baru yaitu sebagai berikut:

- Bowl
- Warped Stem
- Warped Plank
- Furnace
- Crafting Table

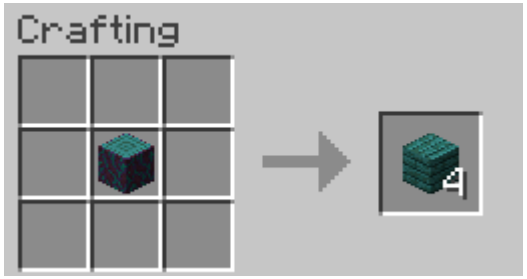
- Oak Door
- Spruce Door
- Birch Door
- Warped Door
- Wooden/Stone/Iron/Diamond Shovel
- Woode/Stone/Iron/Diamond Hoe

Berikut ini adalah resep dari beberapa item diatas

- Bowl

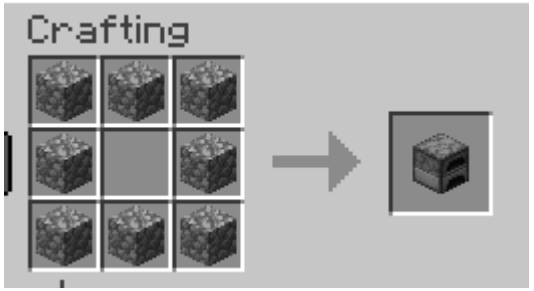
*Jenis Plank dibebaskan

- Warped Plank



Berasal dari Warped Stem

- Furnace



Menggunakan Cobblestone

- Oak Door

Menggunakan Oak Plank

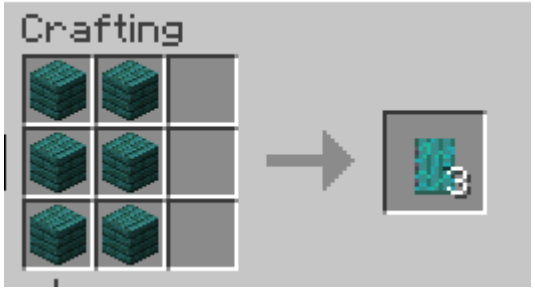
- Spruce Door

Menggunakan Spruce Plank

- Birch Door

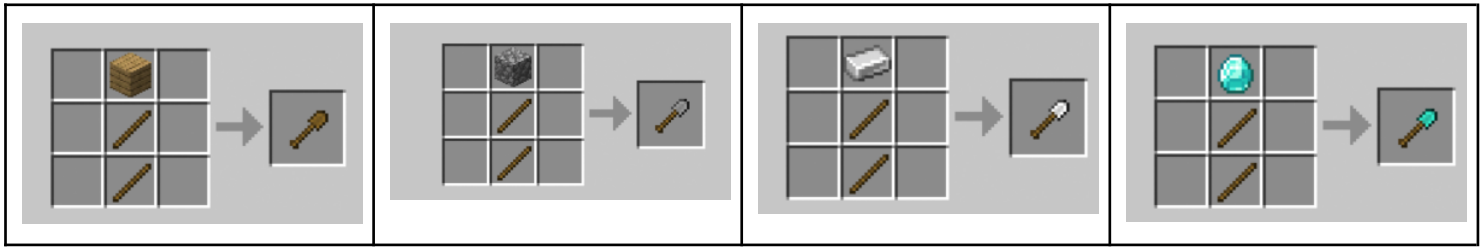
Menggunakan Birch Plank

- Warped Door

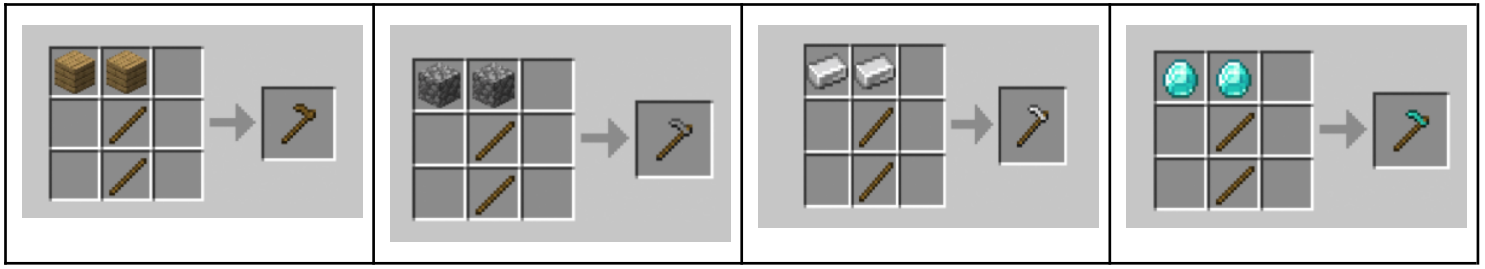


Menggunakan Warped Plank

- Wooden/Stone/Iron/Diamond Shovel



- Wooden/Stone/Iron/Diamond Hoe



3.1.3. Unit Testing Implementation

3.2. Bonus Kreasi Mandiri

3.2.1. Test Case

Untuk menjaga agar setiap perubahan yang dilakukan tidak mengubah hasil, maka diperlukan sebuah testing. Pada saat awal development, kami membuat dua buah jenis tes, yaitu:

- End to end test

  Tes ini dilakukan untuk memeriksa keterhubungan antar kelas dan output dari program. Tes ini diadaptasi dari test runner yang telah termuat pada template. Kami menambahkan fitur disable sehingga untuk tes yang belum terpenuhi dapat dimatikan untuk sementara waktu. Disable dapat dilakukan dengan menyisipkan awalan “DISABLE_” pada fille .ans (File jawaban)

- Unit Test

  Kami menggunakan Google test untuk memeriksa perilaku beberapa kelas yang ada pada program. Hal ini dilakukan untuk mengecilkan peluang untuk terjadinya error disebabkan oleh sebuah kelas.

Test dapat dijalankan dengan menggunakan perintah `make test`.

3.2.2. Help Command

Untuk memudahkan pengguna dalam menggunakan program, disediakan help command untuk menampilkan command apa saja yang tersedia. Berikut adalah cuplikan programnya.

```
#include <command.hpp>

void help(){
    cout << "\nThese are the list of commands and its parameter\n";
    cout << "=======================================================\n";
    cout << "- EXPORT PATH : export inventory to PATH\n";
    cout << "- GIVE ITEMNAME ITEMQTY : Give ITEMQTY of ITEMNAME\n";
    cout << "- MOVE SRCID N DESTID1 DESTID2 ... DESTIDN : Moving N items to the N desired destination, divided equally\n";
    cout << "- MULTIMOVE SRCID N DESTID : Moving N items to one destination destinatio\n";
    cout << "- CRAFT : Crafting current state of crafting table\n";
    cout << "- MULTICRAFT : Crafting current state of crafting table all the way to non-recipe state\n";
    cout << "- DISCARD SRCID ITEMQTY : Delete ITEMQTY items from SRCID\n";
    cout << "- USE SRCID : Use tool item in SRCID\n";
    cout << "- SHOW : Show current inventory and crafting table\n";
    cout << "=======================================================\n";
    cout << "SRCID and DESTID format is collection char concated with slot id, for example I0 for Inventory with slot id 0 as for\n";
    cout << "C10 is crafting table with slot id 10\n\n";
}
```

3.2.3. Multiple Move

Kami juga mengimplementasikan perintah multiple move, yaitu memindahkan N buah item ke satu slot yang tersedia. Berikut adalah cuplikan programnya.

```
3   void doMultiMove(CraftingTable& ct, Inventory& i) {
4     int n;
5     string src, dest;
6
7     cin >> src >> n >> dest;
8     if(n<=0){
9       throw new WrongCommandException(INVALID_PARAMETER);
10    }
11    CraftingTable ctTmp(ct);
12    Inventory iTmp(i);
13
14    vector<string> res;
15    if (src[0] == 'I') {
16      iTmp.moveItem(src, n, vector({dest}), &ctTmp);
17    } else {
18      ctTmp.moveItem(src, n, vector({dest}), &iTmp);
19    }
20
21    ct = ctTmp;
22    i = iTmp;
23    cout << "\nYou successfully moved " << n << " item(s).\n\n";
24  }
```

4. Pembagian Tugas

Modul (dalam poin spek)	Designer	Implementer
Base Exception	13520128	13520128
Program utama	-	13520095
Kelas Slot	13520095	13520095, 13520098, 13520128
Kelas Item	13520053, 13520095	13520053
Kelas Inventory	13520047, 13520095, 13520128	13520047, 13520128
Index Out of Bound Exception	13520047	13520047
Use Non Tool Item Exception	13520053	13520053
Overflow Exception	13520095	13520095
Modul Move Item	13520095	13520095, 13520128
Modul Give Item	13520095	13520095
Modul Crafting	13520095, 13520128	13520095, 13520128
Modul Use	13520098	13520095, 13520098
Modul Show	13520095	13520095
Modul Export	13520095	13520095, 13520128
Modul MultiCraft	13520095	13520095
Modul MultiMove	13520128	13520128

Kelas Recipe	13520128	13520128
Kelas RecipeBook	13520128	13520128
Test Runner: Fitur Disable Test Case	13520128	13520128