Object-Oriented Programming & Class

Modern cpp Programming lecture 3

In this lecture...

- Object-Oriented Programming
- Class
- Other concepts of cpp which needs to understand class

Object-Oriented Programming?

- In the early days of computing…
  - programming == solving problem
  - programming was just the implementation of *algorithms*
  - focused on *the procedure to solve problem*
  - ***Procedural Programming***
    - Low complexity
    - High maintenance
    - Easy to understand

Object-Oriented Programming?

- However, as the complexity of software evolved…
  - Now software is not just algorithms!!
  - It rather became a kind of *simulation*
  - Procedural programming
    - inefficient to *simulate* complex logics
    - components, interactions, hierarchies…
    - => Object-Oriented Programming appeared!!

Object-Oriented Programming

- Simulates real world
- Software is the collection of components (a.k.a. modules)
- Each components is either a logic or ***Object***
- Programmers should define…
  - Objects,
  - their relationships,
  - and the interactions b/w objects

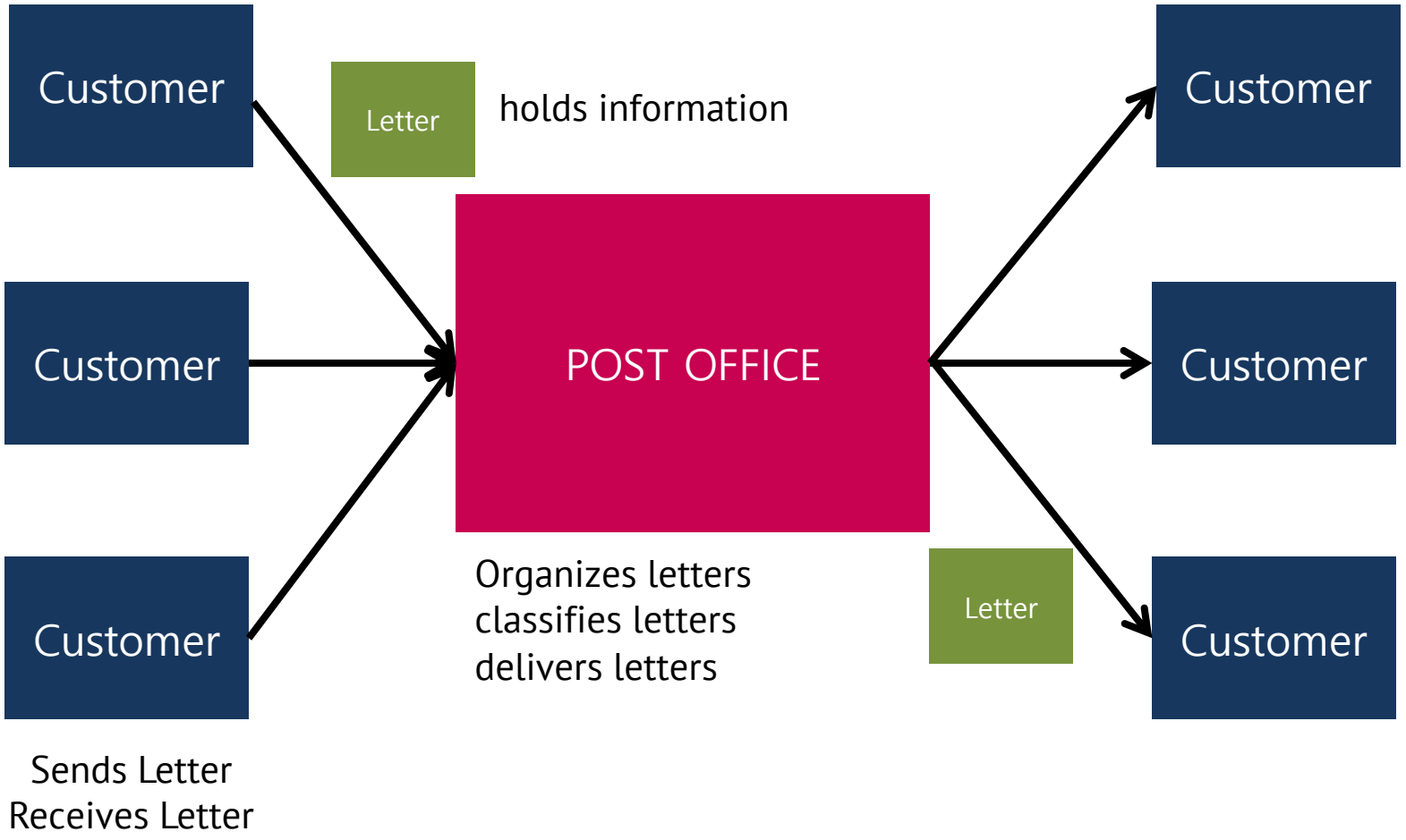
Object-Oriented Programming

- Advantages
  - Most of modern PL => supports OOP
  - Easy Abstraction
  - Supports GUI programming
  - Can easily build *Library* or *module*
    - Easy to reuse code!!
  - Supports Design Pattern (we'll see...)
- Disadvantage
  - slow

Object-Oriented Programming

- Simple example!!
- Suppose you're developing a program…
- which simulates *post office*
- You should design…
  - Post office
  - Customers
  - Letter (or package)
- as objects

Object-Oriented Programming



Object-Oriented Programming

- cpp supports OOP
- Users can define ***classes***
- Class
  - *Class is not a object!!*
  - *Class is just a* ***Data type***
    - like int, float, string, bool…
  - ***User-defined data type***
  - Each class can generate multiple ***objects***

Object-Oriented Programming

- Terminology & Definition
  - Class
    - An extensible program-code-template for creating objects
  - Object
    - In computer science
      - can be a variable, data structure, a function or a method, and as such, is a blue in a value in memory referenced by an identifier.
    - In OOP
      - Refers to a particular instance of a class, where the object can be a com bination of variables, functions, and data structures

- Terminology & Definition
  - Instance
    - concrete occurrence of any object, existing during the runtime of a p rogram
    - is synonymous with *object* as they are each a particular value (realiz ation), and these may be called an *instance object*.
    - Often refers the relationship b/w abstract concept and realization
      - *An Object is the instance of a Class*
      - *A Link b/w Objects is the instance of a relationship b/w Classes*
      - *A Process is the instance of certain Program, or executable*

Class (in cpp)

- Remember!!
  - Defining *new data type!!*
- Data type consists of…
  - value
    - int => integer number
    - string => sequence of characters
  - methods(functions)
    - int => addition, subtraction, multiplication…
    - bool => AND, OR, NOT…

Class

- *User-defined data type* is the same!!
- value
  - *variable*
- functions
  - *methods*
- So many features in class…
- example will help you!!

Class

```
class Human {
private:
    int age;
    string name;
    string nationality;

public:
    Human();
    Human(int age, string name, string nationality);
    ~Human();

    int getAge();
    string getName();
    string getNationality();

    void aging();
    void setName(string newName);
    void setNationality(string newNationality);

    string printPersonalInfo();
    Human* makeProduct(string work);
};
```

variables

methods

Access Identifier

```
class Human {
private:
    int age;
    string name;
    string nationality;

public:
    Human();
    Human(int age, string name, string nationality);
    ~Human();

    int getAge();
    string getName();
    string getNationality();

    void aging();
    void setName(string newName);
    void setNationality(string newNationality);

    string printPersonalInfo();
    Human* makeProduct(string work);

};
```

private and public??

Access Identifier

- Access identifier
  - public
    - accessible from itself / outside the class / child classes
  - private
    - accessible from itself
  - protected
    - accessible from itself / child classes
- child class? we'll see... (in inheritance)
- why access identifier??

Access Identifier

- Why access identifier??
  - if you're working alone…it's okay
  - but if other programmers should use code…
    - collaboration, building library
  - It is better to hide specific information & implementation
  - and just give the concrete way to access class!!
  - ***Information hiding!!***
  - *privatize* information and *publicize* access methods!!

Access Identifier

```
class Human {
private:
    int age;
    string name;
    string nationality;

public:
    Human();
    Human(int age, string name, string nationality);
    ~Human();

    int getAge();
    string getName();
    string getNationality();

    void aging();
    void setName(string newName);
    void setNationality(string newNationality);

    string printPersonalInfo();
    Human* makeProduct(string work);

};
```

direct access to age variable is impossible!!
age can be only changed by aging() method

identifies ownership (class "Human")

```
void Human :: aging() {
    this->age++;
}
```

quite natural...

"this" is the pointer to current object(class instance)

Accessing objects

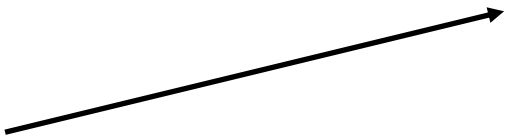
- Object variable can be either pointer or not

```
Human human;
Human* humanPointer = new Human();

human.aging();
humanPointer->aging();
```

- use ”.” if normal variable
- use -> if pointer
- to access it’s element (variable or method)

Getter & Setter

```
class Human {
private:
    int age;
    string name;
    string nationality;

public:
    Human();
    Human(int age, string name, string nationality);
    ~Human();

    int getAge();
    string getName();
    string getNationality();

    void aging();
    void setName(string newName);
    void setNationality(string newNationality);

    string printPersonalInfo();
    Human* makeProduct(string work);

};
```



better to maintain variables as private
Why? publicizing variables
decreases coherency

Therefore, we need getter / setter to access and modify variables!!

Getter & Setter

- get / set : convention

```
void Human :: setName(string newName) {
   this->name = newName;
}

string Human :: getName() {
   return this->name;
}
```

- Always remember!!
- better to privatize variable and publicize it’s getter / setter

Constructor & Destructor

```
class Human {
private:
    int age;
    string name;
    string nationality;

public:
    Human();
    Human(int age, string name, string nationality);
    ~Human();

    int getAge();
    string getName();
    string getNationality();

    void aging();
    void setName(string newName);
    void setNationality(string newNationality);

    string printPersonalInfo();
    Human* makeProduct(string work);

};
```

Constructor

- Constructor
  - literarily “constructs” the class!!
  - constructor name == name of the class
- Default constructor
  - no parameter
  - set default initialization

```
Human :: Human() {
    this->age = 0;
    this->name = "Alice";
    this->nationality = "Korea, Republic of";
}

Human* defaultHuman = new Human();    // dynamic allocation
Human defaultHuman;                   // static declaration
// in both cases, program calls default constructor
```

Constructor

- User-defined Constructor

```
Human :: Human(int age, string name, string nationality) {
    this->age = age;
    this->name = name;
    this->nationality = nationality
}
```

- **“Overloading”!!**
- Same function name, different *signature*
- Users can define various constructor using *function overloading*

Function Overloading

- Different signature -> Different function!!
- signature:
  - function name, argument type, argument number

```
int add(int a, int b) {
   return a + b;
}

int add(int a, int b, int c) {
   return a + b + c;
}

add(1, 2)      // 3
add(1, 2, 3)   // 6
```

  - return type is not the member of signature!!

Copy Constructor

- If you want to *copy* such objects…
- should define & call *copy constructor!!*

```
class Book{
private:
   string title;
   string* authors;

public:
   Book() {
      this->title = "untitled";
      this->authors = new string[3];
      this->authors[0] = "Alice";
      this->authors[1] = "Bob";
      this->authors[2] = "Carol";
   }

   Book(const Book& book) {
      this->title = book.title;
      this->authors = book.authors;
   }
};
```

```
Book book1;
Book book2(book1) // book2 copies book1
```

should follow the form ”const” or “&”??

copy constructor (copies “book”)

Constant variable

- While writing program…
  - some variables should not be modified!!
    - critical features in OS
    - number of commands…
  - However, while collaborating, these variables can be changed!!
  - To prevent such situation, we use const identifier
  - if the program tries to change const variable, it generates error

```
void printConstant(const int a) {
   const string format = "Number: ";
   a = 10;
   format = "Integer";
   cout << format << a << endl;
}
```

compile error!!!!

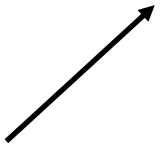
Reference

- Assigning *new name* to the variable!!
- Uses & identifier
- You can access to the variable with any name!!

```
int a = 3;
int& b = a;
cout << a << endl;        // 3
cout << b << endl;        // 3
```

- can also get arguments as references (*call-by-reference)*

```
void modifyInt(int& a) {
   a = 5;
}

int a = 8;
cout << a << endl;        // 8
modifyInt(a);
cout << a << endl;        // 5
```



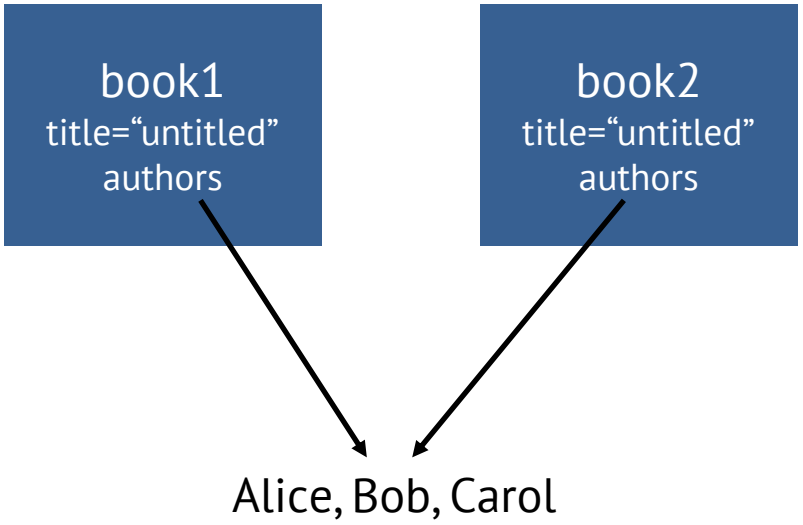
guess why?
You can use reference instead of pointer!!

Copy Constructor

- Okay, finished understanding copy constructor?
- No!!

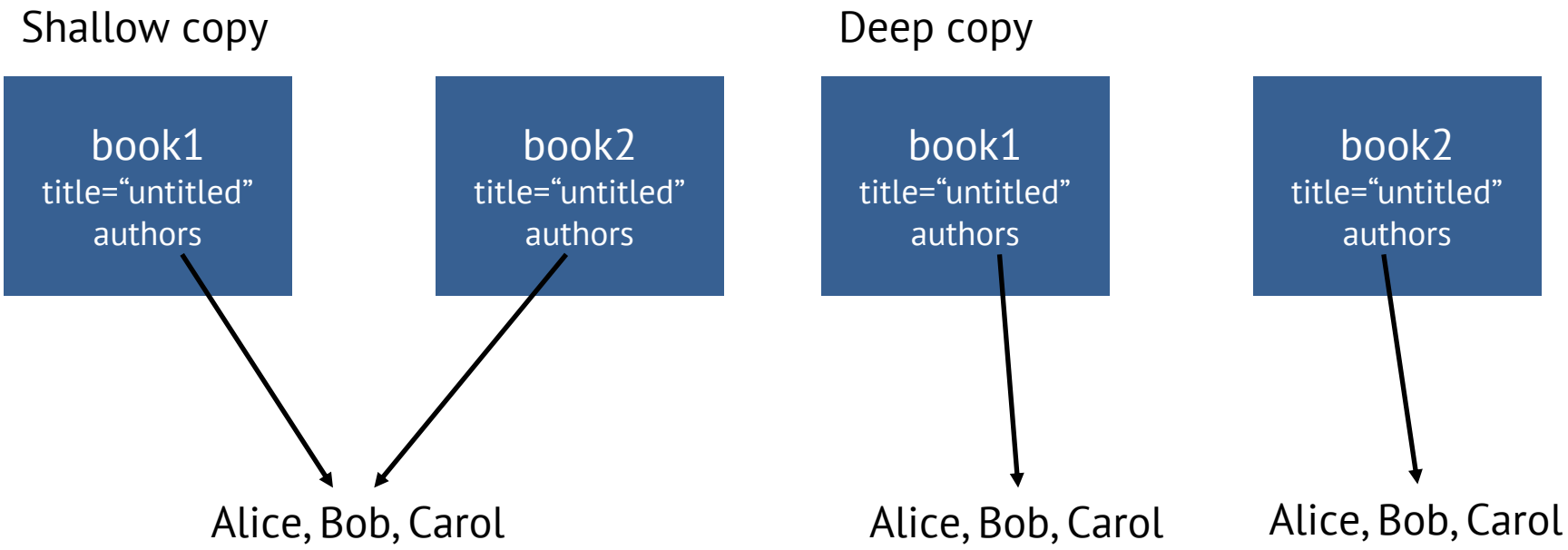
```
class Book{
private:
   string title;
   string* authors;

public:
   Book() {
      this->title = "untitled";
      this->authors = new string[3];
      this->authors[0] = "Alice";
      this->authors[1] = "Bob";
      this->authors[2] = "Carol";
   }

   Book(const Book& book) {
      this->title = book.title;
      this->authors = book.authors;
   }

   void changeFirstAuthor(string author) {
      this->authors[0] = author;
   }
};
```

```
Book book1;
Book book2(book1)
```



If you modify the authors of book1, it also applies to book2

Deep Copy & Shallow copy



should consider the variables inside objects!!

```
Book(const Book& book) {
    this->title = book.title;
    this->authors = new string[3];
    this->authors[0] = book.authors[0];
    this->authors[1] = book.authors[1];
    this->authors[2] = book.authors[2];
}
```

Destructor

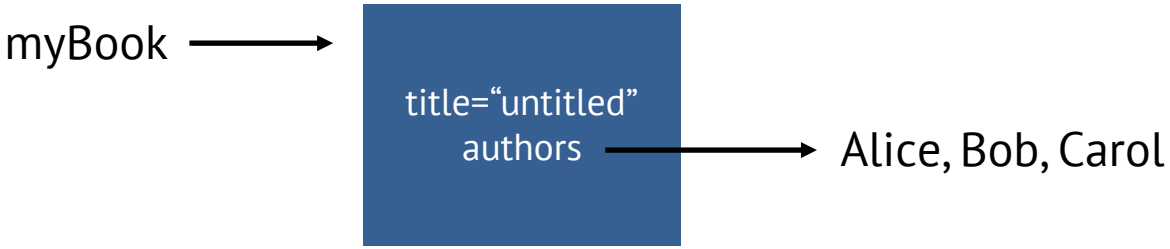
```
class Book{
private:
   string title;
   string* authors;

public:
   Book() {
      this->title = "untitled";
      this->authors = new string[3];
      this->authors[0] = "Alice";
      this->authors[1] = "Bob";
      this->authors[2] = "Carol";
   }

   Book(const Book& book) {
      this->title = book.title;
      this->authors = book.authors;
   }

   ~Book() {
        delete[] authors;
   }
};
```

Destructor

- Calls when the object is destroyed
  - free dynamic allocation
  - end of a function
  - end of a program
- Denoted as ~ + class name
- If the object holds the dynamic-allocated data…
- Destroying object without deallocating the data…
  - is a disaster!!
  - makes dangling memory

Destructor

```
Book* myBook = new Book();
```



if no destructor...default destructor calls!! -> only removes it's variables

```
delete myBook;
```

with destructor, every memory can be freed safely

```
~Book() {
   delete[] authors;
}
```

Class

```
class Human {
private:
    int age;
    string name;
    string nationality;

public:
    Human();
    Human(int age, string name, string nationality);
    ~Human();

    int getAge();
    string getName();
    string getNationality();

    void aging();
    void setName(string newName);
    void setNationality(string newNationality);

    string printPersonalInfo();
    Human* makeProduct(string work);

};
```

Class

- You can define / call any objects and it's method

```
class Human {
private:
   int age;
   string name;
   string nationality;

public:
   ...

   string printPersonalInfo();
   Human* makeProduct(string work);
};

Human :: printPersonalInfo() {
   cout << “Age: ” << age << endl;
   cout << “Name: ” << name << endl;
   cout << “Nationality: “ << nationality << endl;
}

Human human(20, “XiaXia”, ”China”);
human.printPersonalInfo();
```

ASSN2

Election simulator

Election simulator

- Goals
  - understand OOP
  - learn useful skills
    - managing file
    - generate random number
  - Understand pointer & dynamic allocation

Election simulator

- Requirements
  - Should complete three classes:
    - Election
    - District
    - Party
  - Implement the game!!
    - User should select one of two parties (*Together, Integration*)
    - and win the election!!

Election Simulator

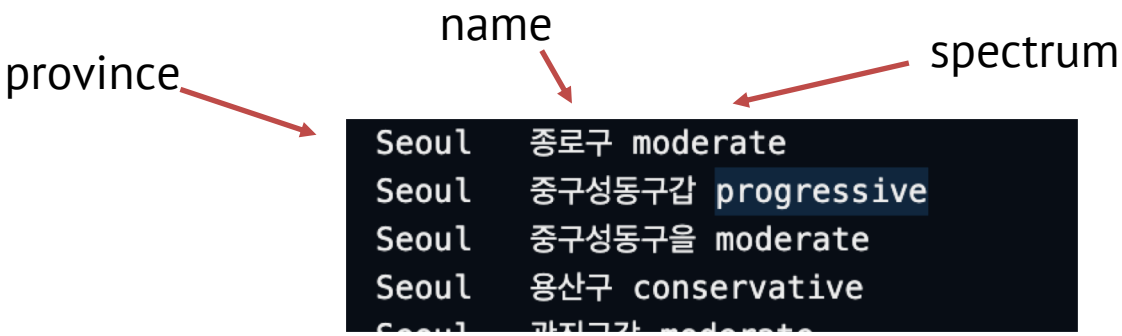
- Party
  - variables
    - name : one of *Together / Integration*
    - isUser : true if user manages the object, otherwise false (AI)
  - If the user selects Together Party, Integration Party automatically becomes AI
  - User should manage the policy of party and win the election!!

Election Simulator

- District
  - variables
    - name : name of the district
    - province : province that the district is belong to
    - spectrum : progressive || moderate || conservative
    - togetherApproval : approval rate of Together Party
  - each district has different spectrum!!
  - Party should control the policy for each spectrum carefully

Election Simulator

- District
  - Get district info from *district_info.txt*
  - Each row has information of a single district



  - Initial approval rate for each spectrum is as follows:
    - progressive: 66 for Together, 33 for Integration
    - moderate: 50 for Together, 49 for Integration
    - conservative: 33 for Together, 66 for Integration

Election Simulator

- Election
  - Class which manages entire simulation
  - single variable
    - `District* district :` the pointer to District Array
    - Dynamic allocation for Districts are already implemented
      - 250 objects for 250 districts information
    - You should read the value in *district_info.txt* and set the information to the allocated District objects.

Election Simulator

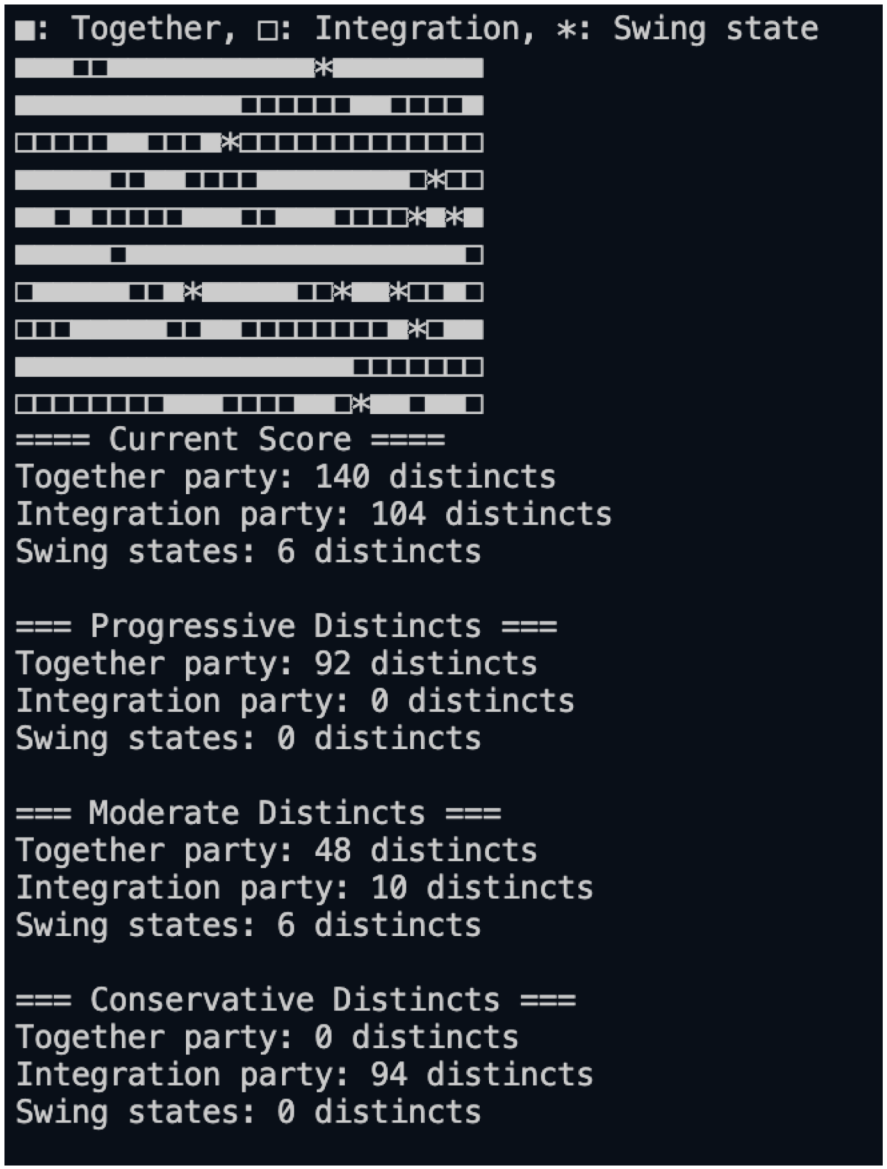
- Member functions (methods)
  - Party
    - Constructor
    - `string getName()`
      - the getter function of name variable
    - `void generatePolicy(int* P, int* M, int* C)`
      - generates policy
      - distributes 60 points to progressive, moderate, conservative districts
      - if the point distributed to certain spectrum is bigger than the one of opposition, approval rate the districts having such spectrum increases
      - If AI party, generates random policy
      - If user party, get input from the user (limit the input which exceeds 60!)

Election Simulator

- Member functions (methods)
  - District
    - you only need to implement basic getter / setter!!
  - Election
    - `printCurrentStatus();`
      - prints current status of the election
      - contents
        - » Election map (provided by helper.cpp's printMap function)
        - » Current score for entire districts
        - » Current score for districts in each spectrum

Election Simulator

- Member functions (methods)
  - Elections
    - `printCurrentStatus();`
    - Swing state
      - if approval rate difference b/w two parties is under 5
      - if togetherApproval >= 52:
        - » Together party
      - elif togheterApproval > 47
        - » Swing state
      - else
        - » Integration party



Election Simulator

```
Possible Provinces:
Seoul, Busan, Incheon, Gwangju
Deajeon, Ulsan, Sejong, Kyungki
Kwangwon, Chungbuk, Chungnam, Jeonbuk
Jeonnam, Kyungbuk, Kyungnam, Jeju
INPUT: Jeonbuk

전주시갑  progressive  T:66, I:33
전주시을  progressive  T:66, I:33
전주시병  progressive  T:66, I:33
군산시  moderate  T:50, I:49
익산시갑  progressive  T:66, I:33
익산시을  progressive  T:66, I:33
정읍시고창군  progressive  T:66, I:33
남원시임실군순창군  moderate  T:50, I:49
김제시부안군  progressive  T:66, I:33
완주군진안군무주군장수군  progressive  T:66, I:33

==== Current Status of Jeonbuk ====
Together party: 8 distincts
Integration party: 0 distincts
Swing states: 2 distincts
```

approval rate

- Member functions (methods)
  - Election
    - `printCurrentProvinceStatus(string province);`
      - get province as parameter
      - prints the info about districts in the province
      - main function (or other function called by main function) should give a parameter
      - If invalid parameter, just return with notification
      - else, prints the info
      - follow the form in above screenshot

Election Simulator

- Member functions (methods)
  - Election
    - `void performStep(int tP, int tM, int tC, int iP, int iM, int iC);`
      - get the policy form each party (Together & Integration) as parameter
      - and perform approval rate update
      - formula:

```
P = tP – iP;
M = tM = iM;
C = tC - iC;
for every progressive districts :
   districts.togetherApproval += P; (with noise)
for every moderate districts :
   districts.togetherApproval += M; (with noise)
for every conservative districts :
   districts.togetherApproval += C; (with noise)
```

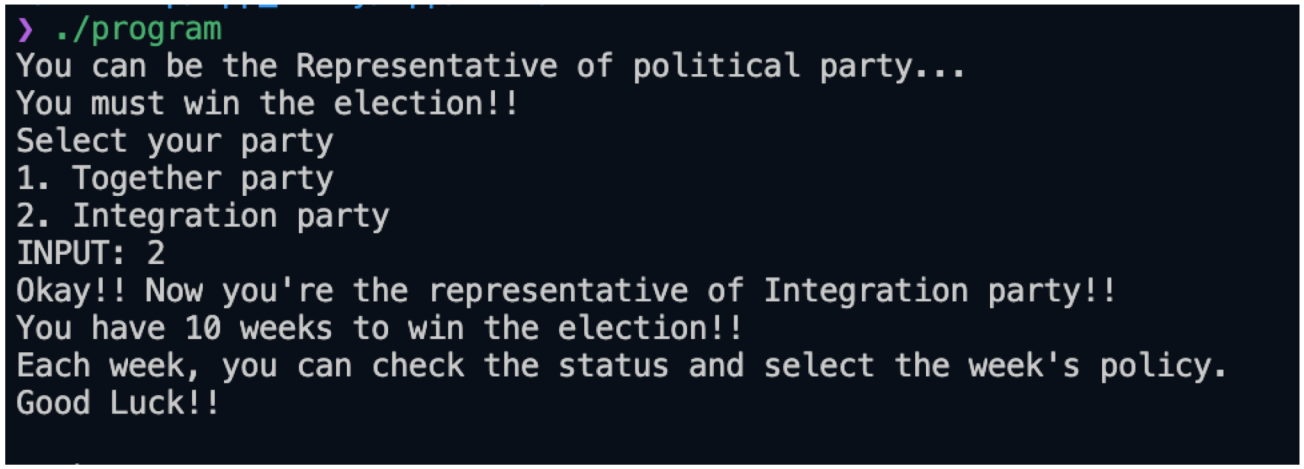
Election Simulator

- Member functions (methods)
  - Election
    - void printFinalResult();
      - notifies the winner of the election!!
      - do not consider swing state

```
==== Final Result ====
Together party: 158 districts
Integration party: 92 districts
Together party WIN!!
```

Election Simulator

- Step
  - Party selection
  - User can select either Together party or Integration Party
  - if invalid input (neither 1 nor 2) : notify and request input again

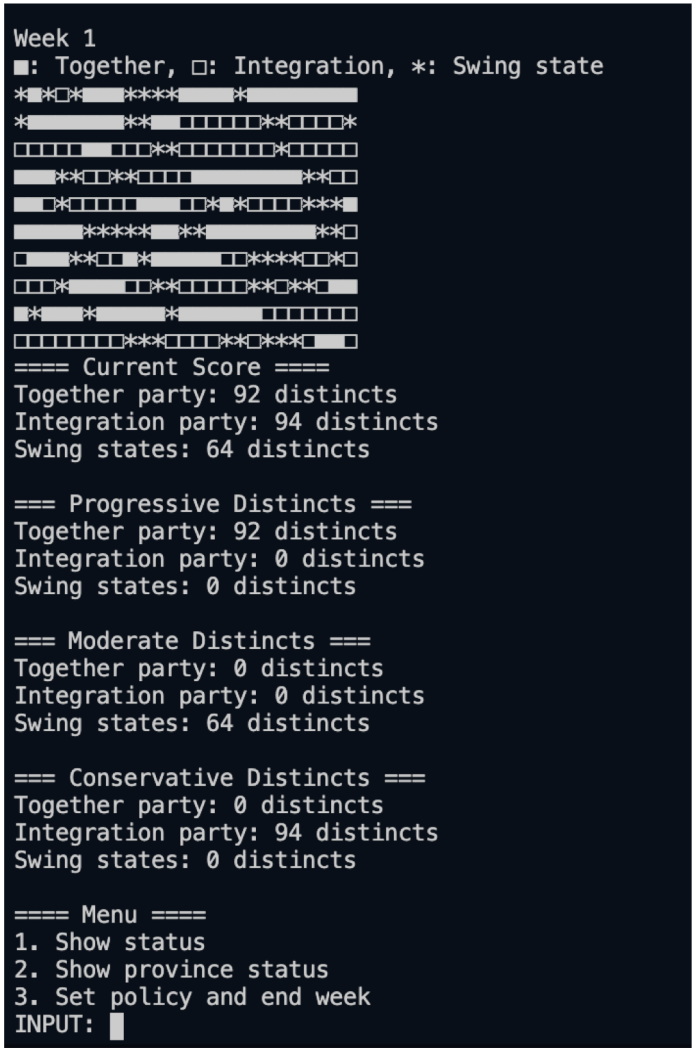


Election Simulator

- Step 2
  - Weekly menu
    - The program first shows the current status, and provides menu
      1. Show status : again visualizes the current status
      2. Show province status : visualizes the current status of certain province
      3. Set policy and end week
    - Able to select 1, 2 repeatedly until the user selects 3.
    - Invalid input => ask input again
  - Total 10 weeks!!

Election Simulator

- Step 2



Invalid input

```
==== Menu ====
1. Show status
2. Show province status
3. Set policy and end week
INPUT: 4
Select again!!
INPUT: 5
Select again!!
INPUT:
```

Province status -> as mentioned before

```
==== Menu ====
1. Show status
2. Show province status
3. Set policy and end week
INPUT: 2

Possible Provinces:
Seoul, Busan, Incheon, Gwangju
Deajeon, Ulsan, Sejong, Kyungki
Kwangwon, Chungbuk, Chungnam, Jeonbuk
Jeonnam, Kyungbuk, Kyungnam, Jeju
INPUT:
```

Election Simulator

- Step 2

```
==== Menu ====
1. Show status
2. Show province status
3. Set policy and end week
INPUT: 3

== Together Party's policy ==
Distribute 60 points to each distinct type!!
Progressive?
```

  - If the user selects 3, asks user the policy for current week
  - You can implement this part freely, but be careful
    - the sum of policy point should not exceed 60!!

Election simulator

- Step 2

```
== Together Party's policy ==
Distribute 60 points to each distinct type!!
Progressive? 30
Moderate? 20
Progressive: 30 points
Moderate: 20 points
Conservative: 10 points

== Integration Party's policy ==
Progressive: 45 points
Moderate: 7 points
Conservative: 8 points

Integration party gained 15 points for progressive districts.
Together party gained 2 points for conservative districts.
Together party gained 13 points for moderate districts.

Input any key to go to next week..
```

- In above example, the user selected Together party, so the program a sks the policy of Together party and automatically generates the one of Integration Party ***(implement the random generation of policy!!)***
- Wait for final input (input any key) to finish week
- Then the program updates the approval value of each province

Election Simulator

- Step 3
  - Approval update
  - using the formula which mentioned before, update the approval rate of the districts

```
P = tP – iP;
M = tM = iM;
C = tC - iC;
for every progressive districts :
   districts.togetherApproval += P; (with noise)
for every moderate districts :
   districts.togetherApproval += M; (with noise)
for every conservative districts :
   districts.togetherApproval += C; (with noise)
```

  - Progress to next week (step 2)
  - ***Noise??***

Election simulator

```
P = tP – iP;
M = tM = iM;
C = tC - iC;
for every progressive districts :
   districts.togetherApproval += P; (with noise)
for every moderate districts :
   districts.togetherApproval += M; (with noise)
for every conservative districts :
   districts.togetherApproval += C; (with noise)
```

- Step 3
  - Noise
    - Every election has unexpected situation!!
    - the program simulates it with “noise”!!
    - for each update, generate random number between -8~8, and add it to P, M, C
  - Example
    - if P is 20, actual update sequence for P will be:
    - 14, 23, 20, 21,14,15,15, 27, 28…

Election Simulator

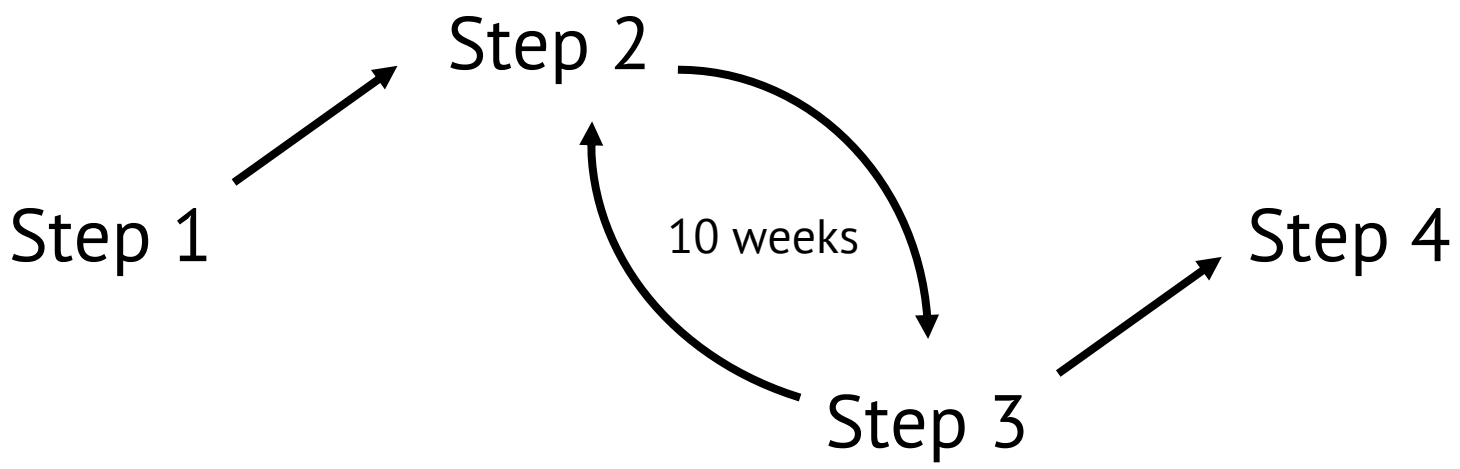
- Step 4
  - Final result
  - After 10 weeks, print the result!!

```
==== Final Result ====
Together party: 158 districts
Integration party: 92 districts
Together party WIN!!
```

  - Then the program ends

Election Simulator

- Program flow



Thank you!!

contact: jeonhyun97@postech.ac.kr