Programming Technique II – SCJ1023

Structured Data

Associate Prof. Dr. Norazah Yusof

What is Abstract Data Types?

- **Abstract Data Types** (ADTs) are data types created by programmer.
- ADTs compose of two groups of elements:
  - a range of data and
  - a set of operations to be performed on the data.
- **Abstraction** is a definition that captures general characteristics of objects without details.

What is data type and structure?

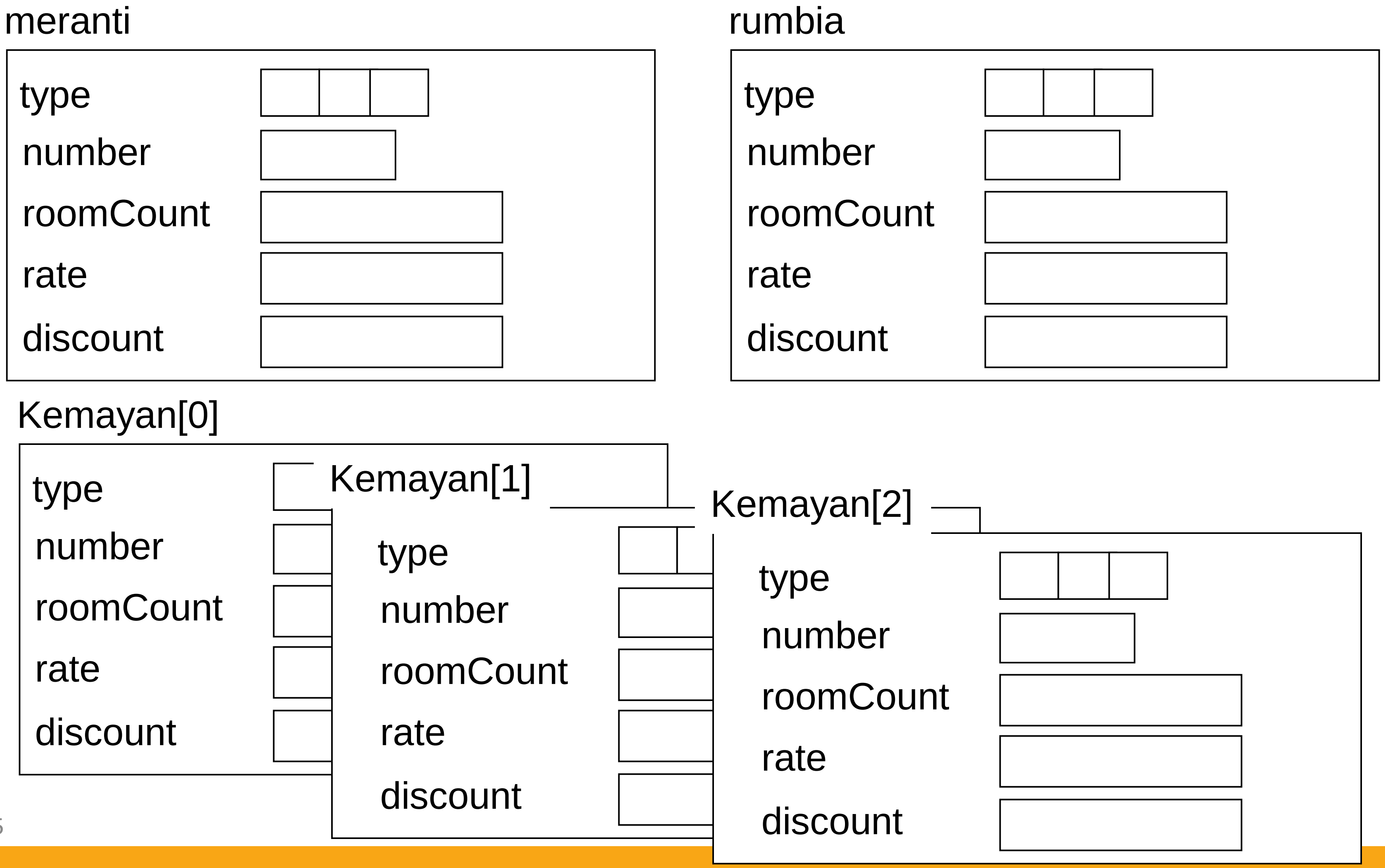
- **Data type** defines the values that can be stored in a variable, for instance, **int**, **char**, **double** and **unsigned long int**.
- **Structure** is a collection of multiple variables into a single name, providing a convenient means of keeping related information together.

Define a structure

- Structure definition does not allocate memory.
- To allocate memory, need to declare a variables of the structure data type.
- Example:

```
struct Chalet {
  char type[3];
  int number;
  int roomCount;
  double rate;
  double discount;
};
Chalet meranti, rumbia, kemayan[3];
```

Memory Layout of variables of type `Chalet`



Accessed a structure member

- Structure members are all variables declarations in a structure.
- Individual members of a structure are accessed through the use of the dot (.) operator.
- Example:

```
 kemayan[2].rate
meranti.number
 rumbia.type[2]
```

Arrays of Structures

- Structures can store several items of varying data types.
- Array of structures can be used to store a list of variable of heterogeneous data types.
- Array of structure definition - same as any other array definition.
- Format:

```
Chalet kemayan[3];
```

Nested Structures

- A structure variable may become a member of another structure variable.
- Example:

```
struct Cost
  {
    double wholesale;
    double retail;
  };
 struct Item
 {
  char partNum[10];
  char description[25];
  Cost pricing;
 };
  Item widget;
```

Pointers to Structures

- A structure variable has an address. Pointers to structures can hold the address of a structure.
- An asterisk is used to declare the pointer variable.
- Operator `&` is used to assign the address
- Example:

```
Cost myCost = {150.00, 200.00};
Cost * costPtr;
 costPtr = &myCost;
```

Accessing Structure Members via Pointer Variables

- Must use () to dereference pointer variable:

```
cout << (*costPtr).wholesale;
```

- Not field within structure:

```
*costPtr.wholesale;
```

- Can use structure pointer operator to eliminate () and use clearer notation:

```
cout << costPtr->wholesale;
```

Deferencing Structure Pointers

- Use the *structure pointer operator*:

```
->
```

A hyphen followed by the greater-than symbol (>).

- Example:

```
costPtr->retail = 350.00;
```

Dynamically Allocating a Structure

- Can use a structure pointer and the *new* operator to dynamically allocate a structure.
- Example to define a `Cost` pointer named `costPtr` and dynamically allocates a `Cost` structure:

```
Cost * costPtr;
costPtr = new Cost;
costPtr->wholesale = 150.00;
costPtr->retail = 250.00;
```

Dynamically Allocating an Array of Structure

- Can also dynamically allocate an array of structures.
- Example to define dynamically an array of five Cost structures, and read the retails of each cost using for loop.

```
Cost * costs;
costs = new Cost[5];
for (int i=0; i< 5; i++)
{
  cout << "Enter the retails for circle " << (i+1) << ": ";
  cin >> circle[i].retails;
}
```