한국외국어대학교
HANKUK UNIVERSITY OF FOREIGN STUDIES

Introduction to Computers & Lab

Lab 12

2021.05.27

Prof. Muhammad Bilal
TA. Sohee Jang

Index

Object

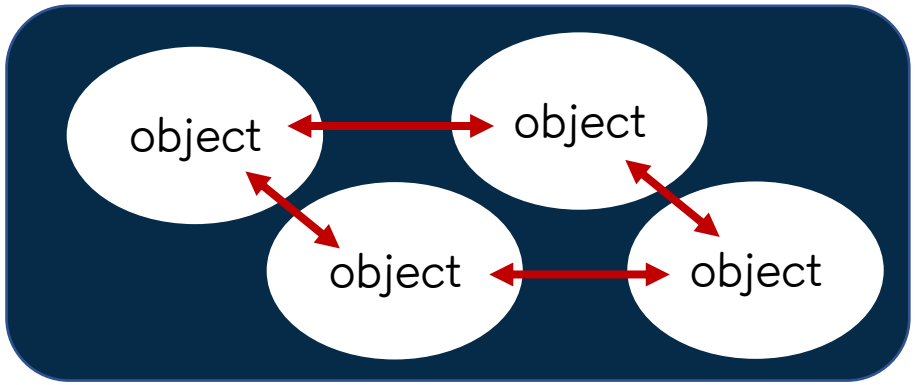
Let's compare the two codes.

```
driveTo(you, work);
```

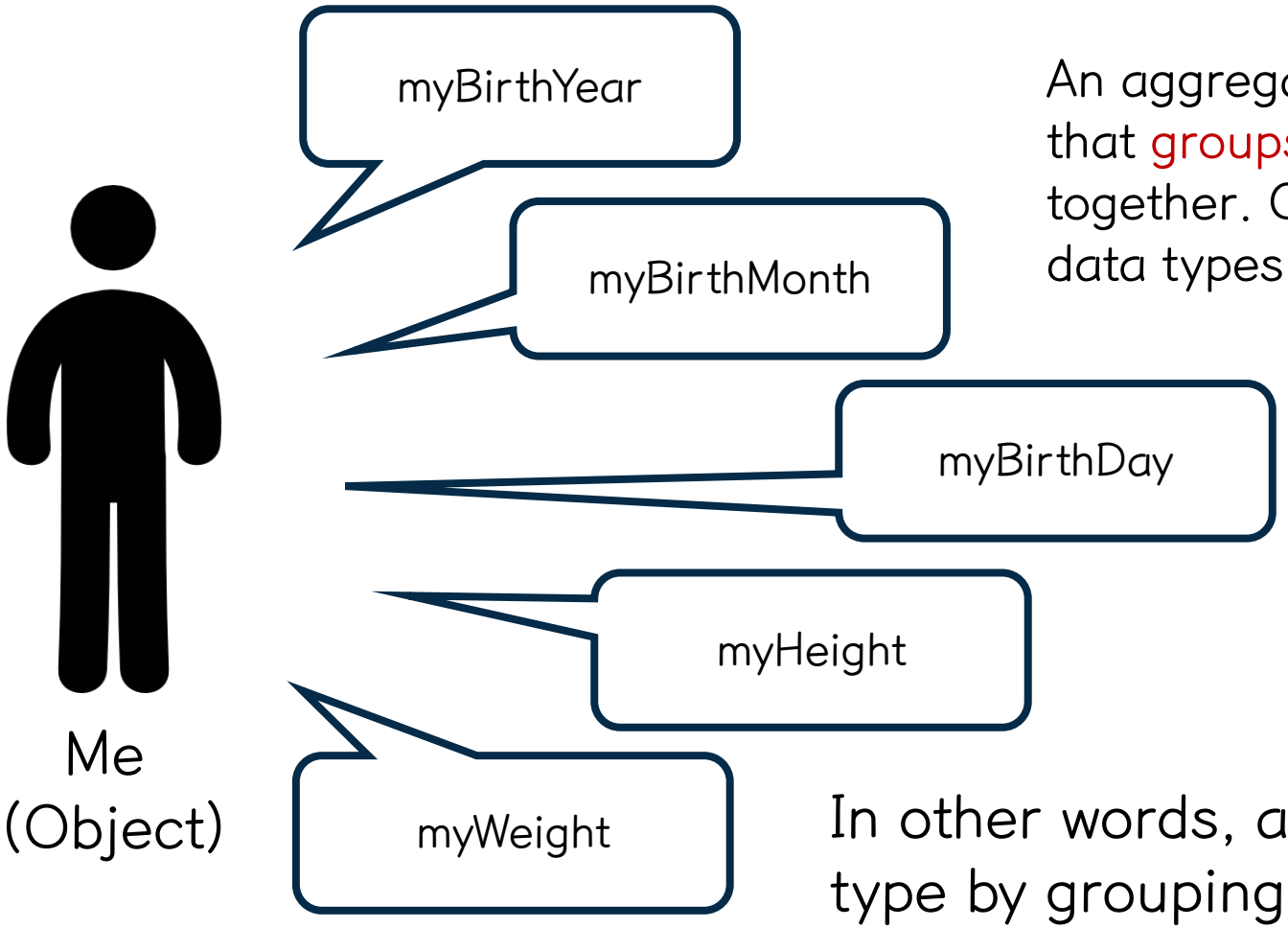
```
you.driveTo(work);
```

The second code can not only be read more clearly, but also clearly know who is the subject and what action will be taken. Rather than focusing on writing functions, we focus on defining objects with clearly defined action sets. This paradigm is therefore referred to as "object-oriented".

program -〉



Structure



An aggregate data type is a type of data that groups multiple individual variables together. One of the simplest aggregate data types is the structure.

In other words, a structure defines a new data type by grouping one or more variables.

Structure

```
struct Employee {
  short id;
  int age;
  double wage;
};
```

member/field

```
Employee joe;
joe.id = 14;
joe.age = 32;
joe.wage = 24.15
```

→ Member selection operator

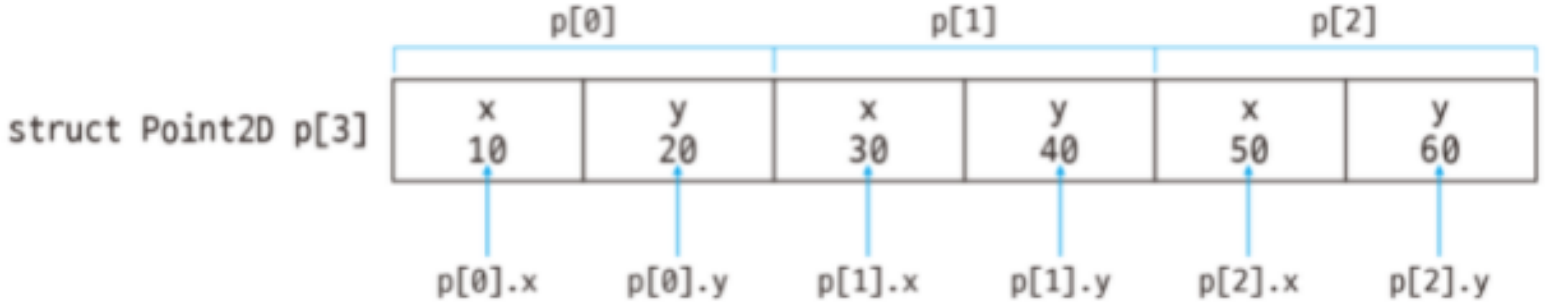
```
Employee frank;
frank.id = 15;
frank.age = 28;
frank.wage = 18.27;
```

* Initializing structs

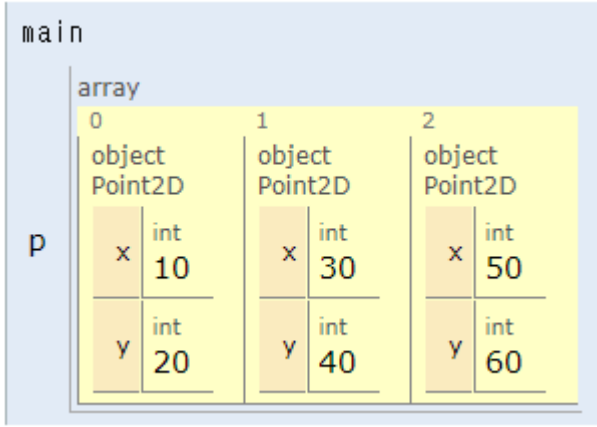
```
Employee joe = {1, 32, 600000.0};
Employee frank = {2, 28};
```

Arrays of Structures

```
#include <iostream>
using namespace std;
struct Point2D {
  int x;
  int y;
};

int main() {
  struct Point2D p[3];

  p[0].x = 10;
  p[0].y = 20;
  p[1].x = 30;
  p[1].y = 40;
  p[2].x = 50;
  p[2].y = 60;

  cout << p[0].x << " " << p[0].y << endl;
  cout << p[1].x << " " << p[1].y << endl;
  cout << p[2].x << " " << p[2].y << endl;

  return 0;
}
```



Arrayname[index].member

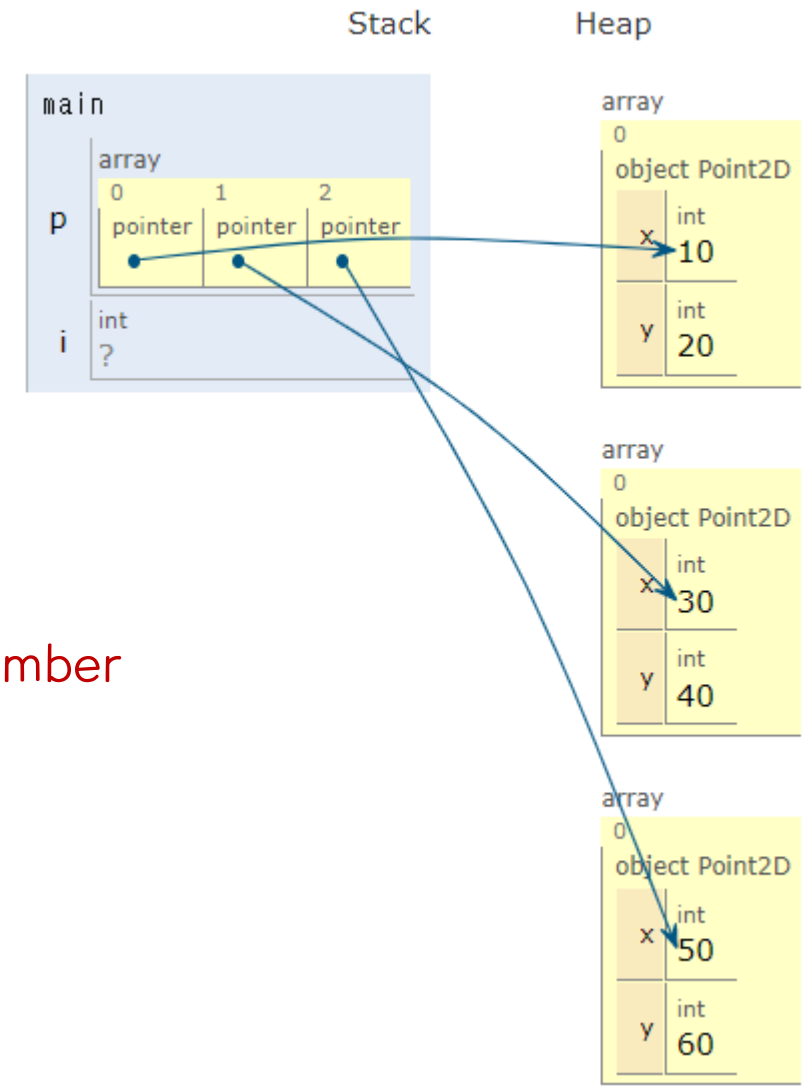



```
10 20
30 40
50 60
```

Pointers to Structure

```
#include <iostream>
#include <stdlib.h>
using namespace std;

struct Point2D {
  int x;
  int y;
};

int main() {
  struct Point2D *p[3];

  for (int i = 0; i < sizeof(p) / sizeof(struct Point2D *); i++) {
    p[i] = (Point2D *)malloc(sizeof(struct Point2D));
  }

  p[0]->x = 10;
  p[0]->y = 20;
  p[1]->x = 30;
  p[1]->y = 40;
  p[2]->x = 50;
  p[2]->y = 60;

  cout << p[0]->x << " " << p[0]->y << endl;
  cout << p[1]->x << " " << p[1]->y << endl;
  cout << p[2]->x << " " << p[2]->y << endl;

  for (int i = 0; i < sizeof(p) / sizeof(struct Point2D *); i++) {
    free(p[i]);
  }

  return 0;
}
```

Arrayname[index]->member



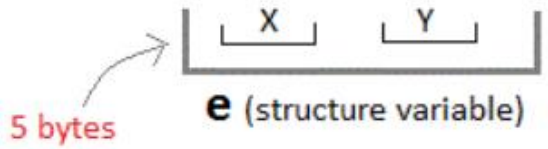

```
10 20
30 40
50 60
```

Union

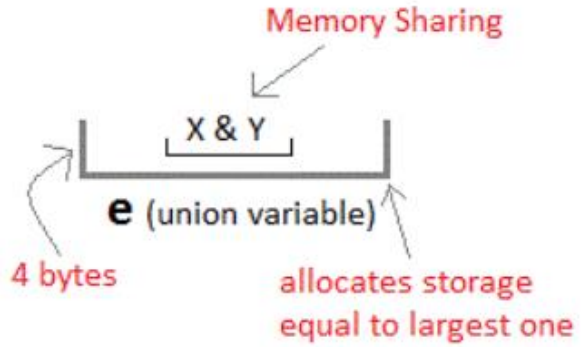
한국외국어대학교
HANKUK UNIVERSITY OF FOREIGN STUDIES

Struct VS Union

```
struct Emp
{
 char X ;     // size 1 byte
 float Y ;    // size 4 byte
} e ;
```



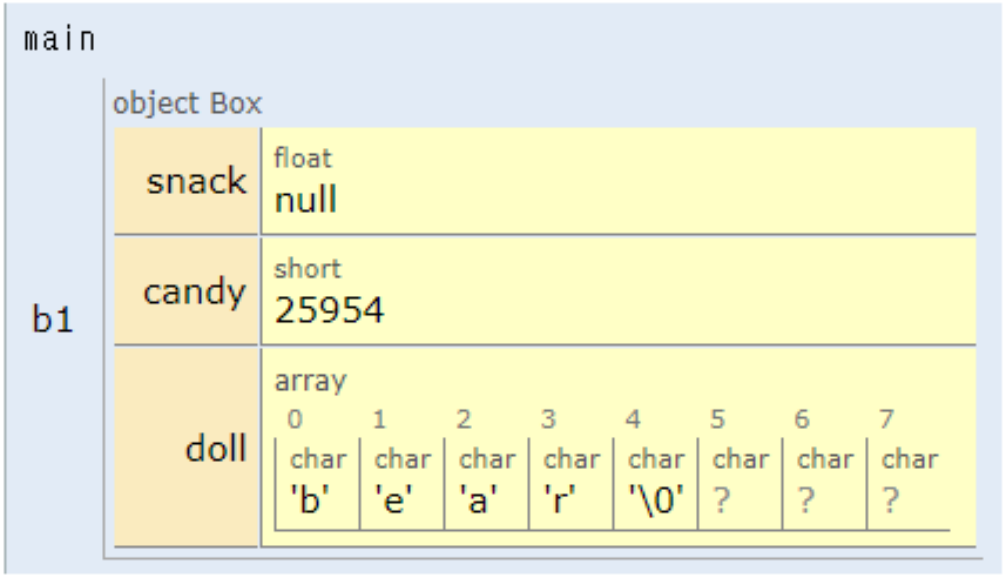

```
union Emp
{
 char X ;
 float Y ;
} e ;
```

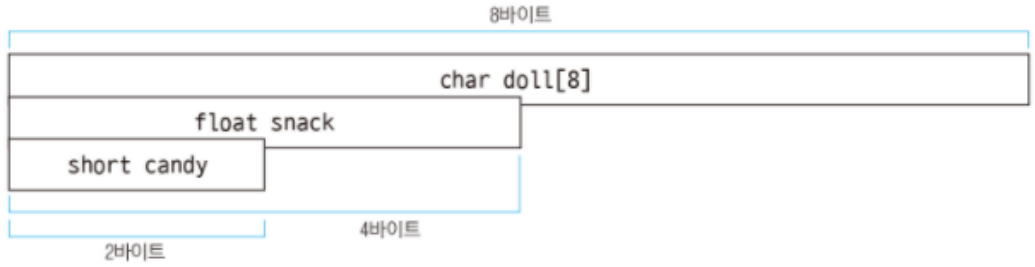


Union

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

union Box {
  short candy; // 2bytes
  float snack; // 4bytes
  char doll[8]; // 8bytes
};

int main() {
  union Box b1;

  cout << sizeof(b1) << endl;

  strcpy(b1.doll, "bear");
  cout << b1.candy << endl;
  cout << b1.snack << endl;
  cout << b1.doll << endl;

  return 0;
}
```

Stack

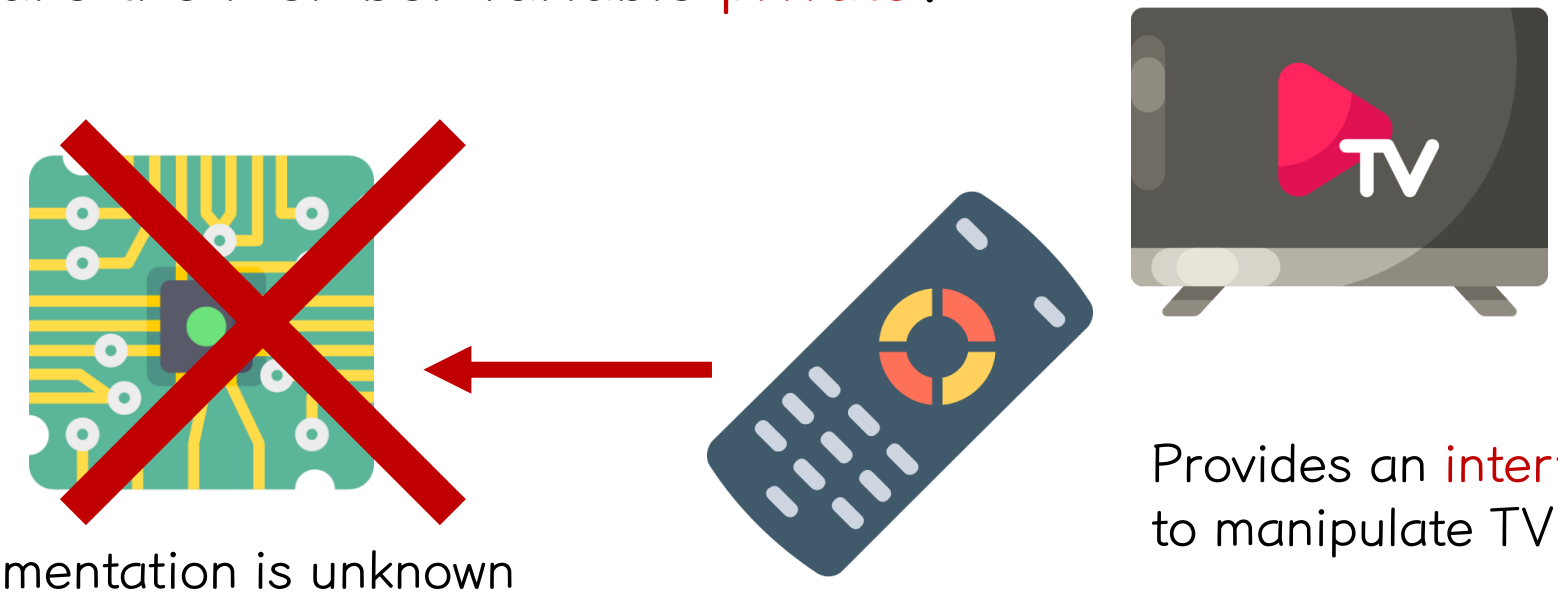





```
8
25954
4.46443e+30
bear
```

Encapsulation

Why do we make the member variable private?



Internal implementation is unknown

Provides an interface to manipulate TV

For similar reasons, implementation and interface separation are very useful in programming.

Encapsulation

한국외국어대학교
HANKUK UNIVERSITY OF FOREIGN STUDIES

In object-oriented programming, encapsulation is a way of keeping details of the way an object is implemented hidden from the user. Instead, the user can access the object through a public interface.

Member variable
(Implementation details) → private

Member function
(public interface) → public
(can be omitted)

Encapsulation

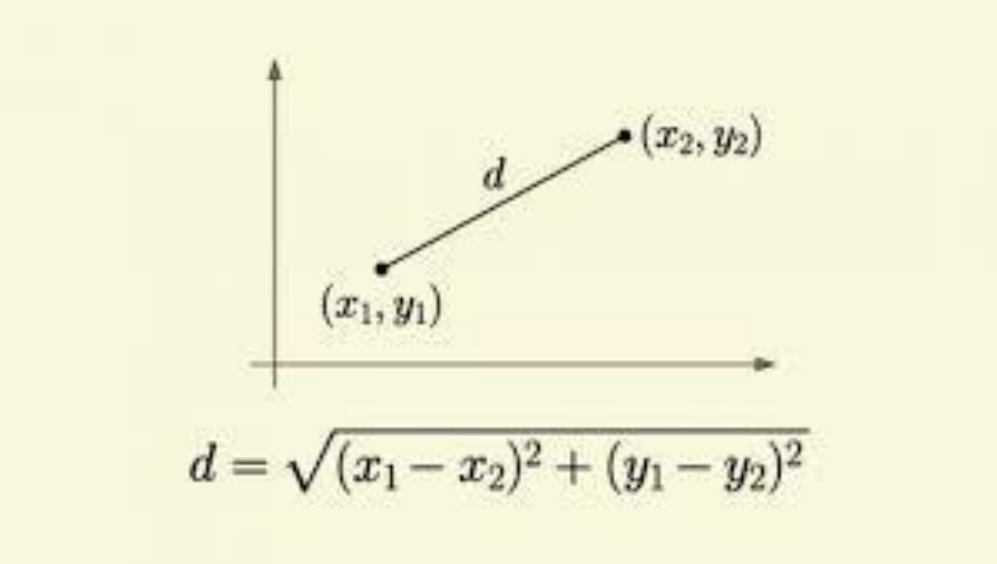

```
struct IntArray {
  public :
    int m_array[10];
}

int main() {
  IntArray array;
  array.m_array[16] = 2;
}
```

```
struct IntArray {
  private :
    int m_array[10];

  public :
    void setValue(int index, int value) {
      if (index < 0 || index >= 10)
        return;

      m_array[index] = value;
    }
}
```

Invalid array index,
now we overwrote memory that we don't own

Task 1 : d(A,B)

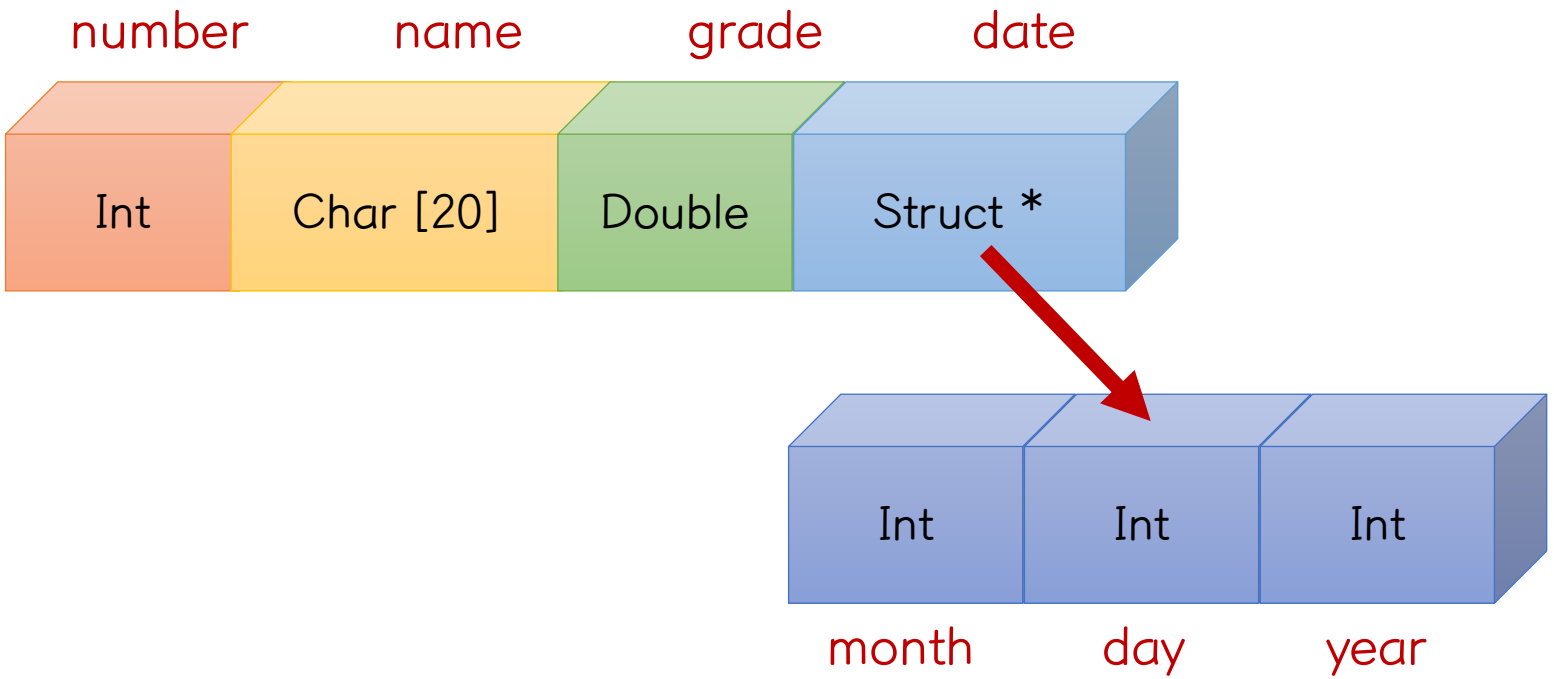
Create a program to determine the distance between two points through a structure representing the x and y coordinates.



★ Call and use the function sqrt() that calculates the square root.

Task 2 : Student list

Declare the structure as shown, and create a program that prints the student's student number, name, grade, and date of birth.



Task 3 : Calories on the menu

한국외국어대학교
HANKUK UNIVERSITY OF FOREIGN STUDIES

For each food, the name of the food and calorie information are expressed in structures. Create a program to represent the foods that appear in each meal course in a structured arrangement and calculate the total calories of each meal course.

Task 4 : Student Info

Create a program that prints student information using a union that stores data about the student, either student ID or resident registration number.

```
#define STU_NUM 1
#define REG_NUM 2

void print(struct student s) {
  switch(s.type) {
    case STU_NUM :
      ....
      break;
    case REG_NUM :
      ....
      break;
    default :
      cout << "type error" << endl;
      break;
  }
}
```

★ Use switch statements to determine the type of union.