Struct and Typedef

Data Structures
C++ for C Coders

한동대학교 김영섭 교수
idebtor@gmail.com

Data Structures

Chapter 2

- ***array, struct, class***
- ***struct and typedef***
- *C string functions*

Arrays

Array: Collections of data of the same type

Why arrays?

- **Efficient random access** (constant time) but **inefficient** insertion and deletion of elements.
- **Good locality** of reference when iterating through - much faster than iterating through (say) a linked list of the same size, which tends to jump around in memory.
- **Consequently,** arrays are most appropriate for storing a fixed amount of data which will be accessed in an unpredictable fashion.

Arrays

Array: Collections of data of the same type

ADT: Array is
objects: A set of pairs **<index, value>** where for each value of index there is value from the set item.
functions:
Array Create (j, list)
Item Retrieve(A, i)
Array Store(A, i, x)

Arrays

Array: Collections of data of the same type

Array example in C:

- **base address:** It is the address of the **first** element of an array which is **&list[0]** or **list**.
- **pointer arithmetic:** (ptr + 1) references to the **next element** of array regardless of its type.
- **dereferencing operator ***
  *(ptr + i) indicates **contents** of the (ptr + i) position of array.

Arrays

Code example: Program 2.1 (modified)

```
void main(void) {
  double array[] = {0, 1, 2, 3, 4};
  int n =

  printf("The sum is: %f\n",  sum(array, n));
  printf("The sum is: %f\n",  sumPointer(&array[0], n));
}
```

equivalent

```
double sum(double a[], int n)
{
  double total = 0;

  for (int i = 0; i < n; i++)
    total += a[i];
  return total;
}
```

```
double sumPointer(double a[], int n)
{
  double total = 0;

  for (int i = 0; i < n; i++)
    total +=
  return total;
}
```

Arrays

Code example: Program 2.1 (modified)

```
void main(void) {
  double array[] = {0, 1, 2, 3, 4};
  int n = sizeof(array) / sizeof(array[0]);

  printf("The sum is: %f\n",  sum(array, n));
  printf("The sum is: %f\n",  sumPointer(&array[0], n));
}
```

equivalent

```
double sum(double a[], int n)
{
  double total = 0;

  for (int i = 0; i < n; i++)
    total += a[i];
  return total;
}
```

```
double sumPointer(double a[], int n)
{
  double total = 0;

  for (int i = 0; i < n; i++)
    total += *a;
  return total;
}
```

i++, a++ 로도 사용가능

Arrays

Code example: Program 2.1 (modified)

```
void main(void) {
  double array[] = {0, 1, 2, 3, 4};
  int n = sizeof(array) / sizeof(array[0]);

  printf("The sum is: %f\n",  sum(array, n));
  printf("The sum is: %f\n",  sumPointer(&array[0], n));
}
```

```
double sum(double a[], int n)
{
  double total = 0;

  for (int i = 0; i < n; i++)
    total += a[i];
  return total;
}
```

```
double sumPointer(double a[], int n)
{
  double total = 0;

  for (int i = 0; i < n; i++)
    total += *a++;
  return total;
}
```

Structures - struct

- **Struct – a handy way to organize data of the different types.**
- Like **class** (actually the idea of class in OOP is derived from **struct**), provide encapsulation of data, it handles a group of data as a whole.
- The **struct** keyword defines a structure type followed by an identifier (name of the structure). Then inside the curly braces, you can declare one or more members of that structure.

C

```
typedef struct Car{
  int  age;
  char tag[32];
}Car;

Car one;
one.age = 21;
strcpy(one.tag, "sky");
```

Structures - typedef

- The **typedef** is used to give a data type a new name.
  It is mostly done in order to make the code cleaner.

- Keyword **typedef** can be used to simplify syntax of a structure in C.
- In C++, you can do the same thing without typedef and more.

C

```
typedef struct Car{
  int  age;
  char tag[32];
}Car;

Car one;
one.age = 21;
strcpy(one.tag, "sky");
```

C++

```
struct Car {
  int    age;
  string tag;
};

Car one;
one.age = 21;
one.tag = "sky";
```

struct as a type and pointer

- Recall a pointer can store only a address of memory.

C++

```
struct Car{
    string tag;
    int    age;
};

Car  ur = {"cat", 25};
Car* my = (Car *)malloc(sizeof(Car));
```

struct as a type and pointer

- Recall a pointer can store only a address of memory.

C++

```
struct Car{
    string tag;
    int    age;
};

Car  ur = {"cat", 25};
Car* my = (Car *)malloc(sizeof(Car));
```

C++

```
struct Car{
    string tag;
    int    age;
};

Car  ur = {"cat", 25};
Car* my = new Car {"sky", 20};
```

new가 있으면 반드시 포인터 >> 메모리 얼로케이션 후 주고를 받아야하기 때문에

struct as a type and pointer

- Recall a pointer can store only a address of memory.

C++

```
struct Car{
    string tag;
    int    age;
};

Car  ur = {"cat", 25};
Car* my = new Car {"sky", 20};
my.tag = "joy";
my.age = 15;
```

struct as a type and pointer

- Recall a pointer can store only a address of memory.

C++

```
struct Car{
    string tag;
    int    age;
};

Car  ur = {"cat", 25};
Car* my = new Car {"sky", 20};
```

C++

```
struct Car{
    string tag;
    int    age;
};

Car  ur = {"cat", 25};
Car* my = new Car {"sky", 20};
my.tag = "joy";
my.age = 15;
```

wrong!

struct as a type and pointer

- Recall a pointer can store only a address of memory.

C++

```
struct Car{
    string tag;
    int    age;
};

Car  ur = {"cat", 25};
Car* my = new Car {"sky", 20};
(*my).tag = "joy";
(*my).age = 15;
```

C++

```
struct Car{
    string tag;
    int    age;
};

Car  ur = {"cat", 25};
Car* my = new Car {"sky", 20};
my.tag = "joy";
my.age = 15;
```

wrong!

struct as a type and pointer

- Recall a pointer can store only a address of memory.

C++

```
struct Car{
    string tag;
    int    age;
};

Car  ur = {"cat", 25};
Car* my = new Car {"sky", 20};
(*my).tag = "joy";
(*my).age = 15;
```

C++

```
struct Car{
    string tag;
    int    age;
};

Car  ur = {"cat", 25};
Car* my = new Car {"sky", 20};
my->tag = "joy";
my->age = 15;
```

better!

struct as a type and pointer

- Passing a pointer to a function.

C++

```
bool older(Car *a, Car *b) {
    return a->age > b->age ;
};

int main() {
  Car  ur = {"cat", 25};
  Car* my = new Car {"sky", 20};

  bool ans = older(______, ______);
}
```

struct as a type and pointer

❖ Passing a pointer to a function.

C++

```
bool older(Car *a, Car *b) {
    return a->age > b->age ;
};

int main() {
  Car  ur = {"cat", 25};
  Car* my = new Car {"sky", 20};

  bool ans = older(&ur, my);
}
```

❖ **Don't you see a bug above?**

struct as a type and pointer

- Passing a pointer to a function.

C++

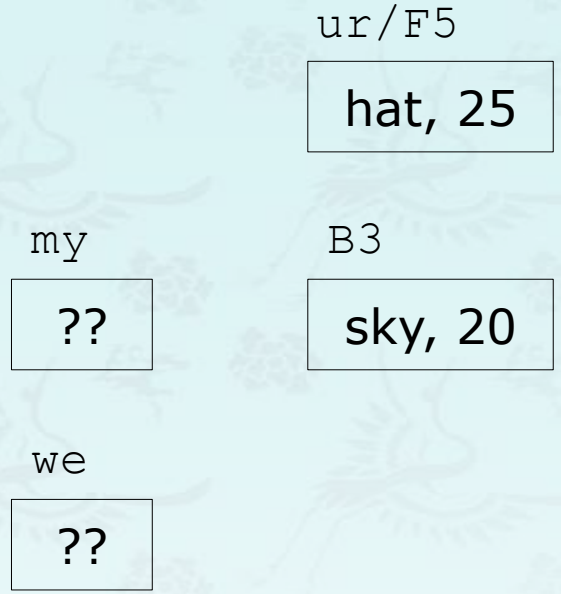
```
bool older(Car *a, Car *b) {
    return a->age > b->age ;
};

int main() {
  Car  ur = {"cat", 25};
  Car* my = new Car {"sky", 20};

  bool ans = older(&ur, my);
  delete my;
}
```

Quiz

1. **Complete the diagram**
2. **Fix an error in the code.**
3. **What is the output of the code ?**



C++: an error in the code

```
int main() {
  Car  ur = {"cat", 25};
  Car* my = new Car {"sky", 20};
  Car* we = &ur;
  ur.tag = "hat";
  cout << we.tag << endl;
  delete my;
}
```

struct as a type and pointer

Q: How copying **my** car to **ur** car?

C++

```
struct Car{
    string tag;
    int    age;
};

Car  ur = {"cat", 25};
Car* my = new Car {"sky", 20};

(1) ur = *my;
(2) ur = my;
(3) ur = &my;
(4) *ur = my;
```

struct as a type and pointer

Q: How copying **my** car to **ur** car?

C++

```
struct Car{
    string tag;
    int    age;
};

Car  ur = {"cat", 25};
Car* my = new Car {"sky", 20};

ur = *my;
```

struct as a type and pointer

- Let's go one more step!
  How about redefining **Car* my** since we are going to love the pointer?

C++

```
struct Car{
    string tag;
    int    age;
};

Car  ur = {"cat", 25};
Car* my = new Car {"sky", 20};

ur = *my;
```

C++

```
struct Car{
    string tag;
    int    age;
};
using pCar = Car*;
Car  ur = {"cat", 25};
pCar my = new Car {"sky", 20};

ur = *my;
```

ECE 20010 Data Structures

Data Structures

Chapter 2

- ***array vs. struct***
- ***struct and typedef***
- *C string functions*

Prof. Youngsup Kim, idebtor@handong.edu, Handong Global University


Summary

&


quaestio quaestio qo ? ? ? ?