Object oriented programming In C++ (ACE 1004)

Structure 2

Prof. 최학남

xncui@inha.ac.kr

Office: high-tech 401

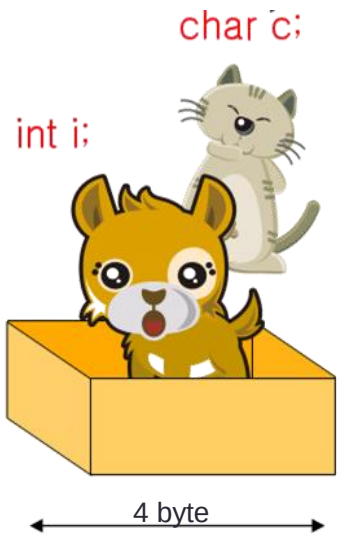
Contents

- Definition of **structure**
- Declaration of **structure**
- Initialization of **structure**
- Access the **structure**

Union

- A union, is a collection of variables of different types, just like a structure. However, with unions, you can only store information in one field at any one time.

```
union example {
        char c;                 // share the same memory space
        int i;                  // share the same memory space
};
```



EX #5: union

```
#include <stdio.h>

union example {
      int i;
      char c;
};

int main(void)
{
      union example v;

      v.i = 100;
      printf("v.c:%c   v.i:%i\n", v.c, v.i );

      v.c = 'A';
      printf("v.c:%c   v.i:%i\n", v.c, v.i);

}
```

Declaration of union

Declaration of union variable

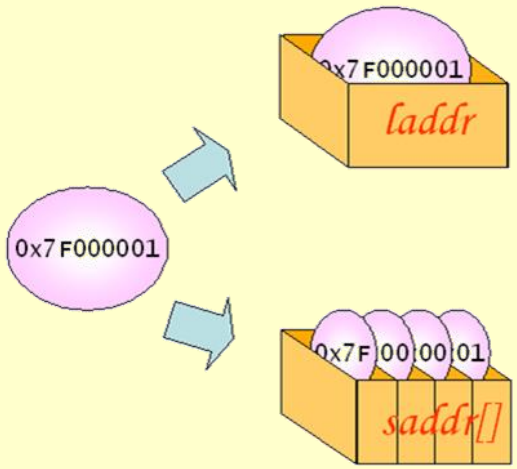
refer the int type

Refer the char type

```
v.c:d  v.i:100
v.c:A v.i:65
```

EX#6: ip address

```
#include <stdio.h>

union ip_address {
      unsigned long laddr;
      unsigned char saddr[4];
};

int main(void)
{
      union ip_address addr;

      addr.saddr[0] = 1;
      addr.saddr[1] = 0;
      addr.saddr[2] = 0;
      addr.saddr[3] = 127;

      printf("%x\n", addr.laddr);

      return 0;
}
```



7f000001

EX#7 : Same type field

```
#include <stdio.h>

#define STU_NUMBER 1
#define REG_NUMBER 2

struct  student {
        int type;
        union {
                int stu_number;             // Student ID (SID)
                char reg_number[15];        // Registration ID (RID)
        } id;
        char name[20];
};
void print(struct student s)
{
        switch(s.type)
        {
                case STU_NUMBER:
                        printf("SID: %d\n", s.id.stu_number);
                        printf("Name: %s\n", s.name);
                        break
                case REG_NUMBER:
                        printf("RID: %d\n", s.id.reg_number);
                        printf("Name: %s\n", s.name);
                        break
                default:
                        printf("type error\n");
                        break
        }
}
```

EX#7 : Same type field (Cont.)

```
int main(void)
{
	struct student s1, s2;

	s1.type = STU_NUMBER;
	s1.id.stu_number = 20070001;
	strcpy(s1.name, "Hong");

	s2.type = REG_NUMBER;
	strcpy(s2.id.reg_number, "860101-1058031");
	strcpy(s2.name, "Kim");

	print(s1);
	print(s2);

	return 0;
}
```

```
SID: 20070001
Name: Hong
RID: 860101-1058031
Name: Kim
```

Enumeration (enum)

- **enum** is a user-defined type consisting of a set of enumerators(enumerator - named integer constant)
- keyword : *enum*

```
enum tag_name { value 1, value 2, ... };
```

```
enum days1 { MON, TUE, WED, THU, FRI, SAT, SUN };
enum days2 { MON=1, TUE, WED, THU, FRI, SAT, SUN };
enum days3 { MON=1, TUE=2, WED=3, THU=4, FRI=5, SAT=6, SUN=7 };
enum days4 { MON, TUE=2, WED=3, THU, FRI, SAT, SUN };

enum days1 d;
d = WED;
```

EX #8 : Enumeration

```
enum days  { SUN, MON, TUE, WED, THU, FRI, SAT };

enum colors { white, red, blue, green, black };

enum boolean { 0, 1 };

enum months { JAN, FEB, MAR, APR, MAY, JUN, JUL, AUG, SEP, OCT, NOV, DEC };

enum major { COMMUNICATION, COMPUTER, ELECTRIC, ELECTRONICS };

enum component { MAIN_BOARD, CPU, GRAPHIC_CARD, DISK, MEMORY };

enum levels { low = 1, medium, high };

enum CarOptions
{
   SunRoof = 0x01,
   Spoiler = 0x02,
   FogLights = 0x04,
   TintedWindows = 0x08,
}
```

EX 9: enum type

```
#include <stdio.h>
enum tvtype { LCD, LED, PDP, TD };

int main(void)
{
	enum tvtype type;

	printf("Enter the tpye of TV : ");
	scanf("%d", &type);
	switch(type)
	{
		case LCD:
			printf("LCD TV.\n");
			break;
		case LED:
			printf("LED TV.\n");
			break;
		case PDP:
			printf("PDP TV.\n");
			break;
		case TD:
			printf("3D TV.\n");
			break;
		default:
			printf("reselect the TV Type.\n");
			break;
	}
	return 0;
}
```

```
Enter the type of TV : 2
PDP TV.
```

Comparison between enum and other methods

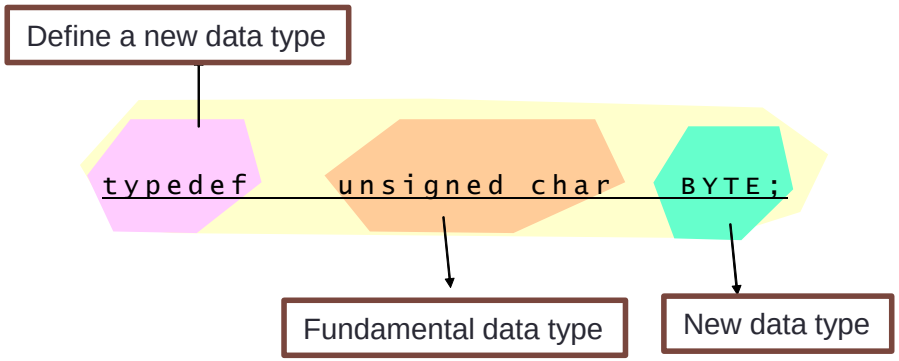
Int	#define	Enum

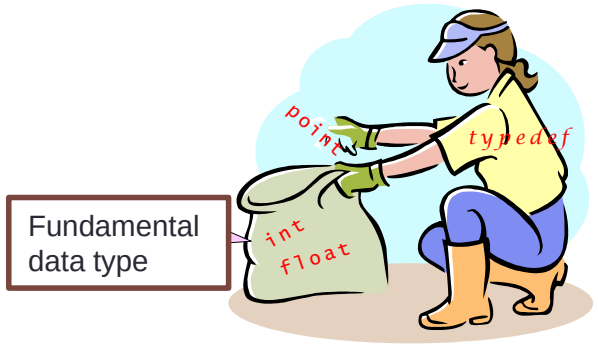
switch(code) {
  case 1:
  printf("LCD TV\n");
  break;
  case 2:
  printf("PDP TV\n");
  break;
}
	#define LCD 1
	#define PDP 2
	
	switch(code) {
	  case LCD:
	  printf("LCD TV\n");
	  break;
	  case PDP:
	  printf("PDP TV\n");
	  break;
	}
		enum tvtype { LCD, PDP };
		enum tvtype code;
		
		switch(code) {
		  case LCD:
		  printf("LCD TV\n");
		  break;
		  case PDP:
		  printf("PDP TV\n");
		  break;
		}

typedef

- Typedef: define a new data type
- Extend the fundamental data type

```
typedef    old_type    new_type;
```





Example of typedef

Fundamental data type	Redefined data type
int	INT32
short	INT16
unsigned int	UINT32
unsigned short	UINT16
unsigned char	UCHAR, BYTE
char	CHAR

```
typedef int  INT32;
typedef unsigned int  UINT32;

INT32 i;                    // same as int i;
UINT32 k;                   // same as unsigned int k;


typedef struct point {
                int x;
                int y;
} POINT;

POINT p,q;
```

```
typedef struct complex {
                double real;
                double imag;
} COMPLEX;

COMPLEX x, y;

typedef enum { FALSE, TRUE } BOOL;

BOOL condition;             // enum { FALSE, TRUE }
condition;

typedef char * STRING_PTR;

STRING_PTR p;               // char *p;
```

EX #10

```c
#include <stdio.h>

typedef struct point {
	int x;
	int y;
} POINT;

POINT translate(POINT p, POINT delta);

int main(void){
	POINT p = { 2, 3 };
	POINT delta = { 10, 10 };
	POINT result;

	result = translate(p, delta);
	printf("coordinate of the new point: (%d, %d)\n", result.x, result.y);

	return 0;
}

POINT translate(POINT p, POINT delta){
	POINT new_p;

	new_p.x = p.x + delta.x;
	new_p.y = p.y + delta.y;

	return new_p;
}
```

Coordinate of the new point (12, 13)

EX #11 Size(memory) checking(32bit OS)

- Union

```
union test{
int a;
double b;
char name[9];
};
```

16byte

```
struct test1 {
double a;
char b;
short c;
int d;
};
```

16byte

```
struct test2 {
char a;
double b;
short c;
int d;
};
```

24byte

EX #12 Size(memory) checking

1,2,4,8,16

- Union `#pragma pack(1)`

```
union test{
int a;
double b;
char name[9];
};
```

9byte

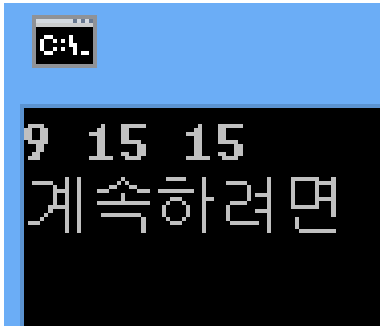
```
struct test1 {
double a;
char b;
short c;
int d;
};
```

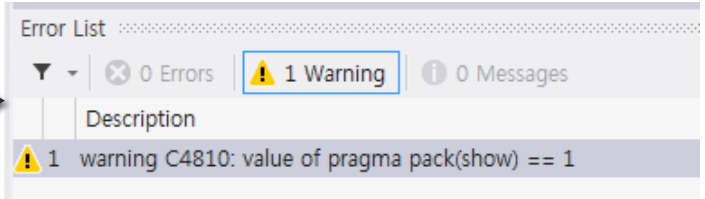
15byte

```
struct test2 {
char a;
double b;
short c;
int d;
};
```

15byte

```
#include <iostream>
using namespace std;
#pragma pack(1)
#pragma pack(show)
union id {
    int a;
    double b;
    char name[9];
};
struct test1 {
    double a;
    char b;
    short c;
    int d;
};
struct test2 {
    char a;
    double b;
    short c;
    int d;
};
void main()
{
    id i={0};
    test1 t1={0};
    test2 t2={0};
    cout<<sizeof(i)<<" "<<sizeof(t1)<<" "<<sizeof(t2)<<endl;
}
```





EX #13 Size(memory) checking

- Union

```
#pragma pack(4)
union test{
int a;
double b;
char name[9];
};
```

?byte

```
struct test1 {
double a;
char b;
short c;
int d;
};
```

?byte

```
struct test2 {
char a;
double b;
short c;
int d;
};
```

?byte

HW#8

1. Define three student variables. Read the scores for the three students and display the score report.

Enter scores for 3 students

student 1: 80 90 70

student 2: 40 30 30

student 3: 90 90 90

The score report

student 1

korean: 80 english: 90 math: 70 total: 240

student 2

.....

HW#8

2. Same as 1), but extend the student struct to include id, age and sex for each student.

Enter student ID, age, sex(M for man, F for woman), and scores for 3 students
student 1: 12345 19 M 80 90 70
student 2: 12346 20 F 40 30 30
student 3: 11223 22 M 90 90 90

The score report
student 1
student ID: 12345 age: 19 sex:M
korean: 80 english: 90 math: 70 total: 240
student 2:
student ID: 12346 age: 20 sex:F
korean: 40 english: 30 math: 30 total: 100
.....

HW#8

3. Same as 2), but add name to each student. Use string for the name.

enter student name, ID, age, sex(M for man, F for woman), and korean, english, math scores for 3 students

student 1: kim 12345 19 M 80 90 70
student 2: park 12346 20 F 40 30 30
.....
The score report
student 1
student name: kim student ID: 12345 age: 19 sex:M
korean: 80 english: 90 math: 70 total: 240
student 2:
student name: park student ID: 12346 age: 20 sex:F
korean: 40 english: 30 math: 30 total: 100
.....

HW #8

4.1. Try the given code.

4.2. Implement student data management system. Define the “student array” to global.

```
1. load  2. show  3. best score  4. male students  5. female students  6. change
select menu
1
enter data for 5 students (name id sex korean english math scores)
.................
1. load  2. show  3. best score  4. male students  5. female students  6. change
select menu
4
The data for male students
.............
1. load  2. show  3. best score  4. male students  5. female students  6. change
select menu
3
best score for korean: kim 99
best score for english: park 97
best score for math: park 90
1. load  2. show  3. best score  4. male students  5. female students  6. change
select menu
6
enter student ID
11233
enter new data(name id sex korean english math)
……..
```

HW #8

Given code:

```
struct student{
   char name[20];
   int id;
   char sex;
   int kor, eng, math;
};
void main(){
   struct student  std[5];  // 5 students
   int i;
    printf("enter name, id, sex, kor, eng, math for 5 students\n");
   for(i=0;i<5;i++){  // read data for each student
      printf("student %d: ", i);
      scanf("%s %d %c %d %d %d", &std[i].name, &std[i].id, &std[i].sex,
                              &std[i].kor, &std[i].eng, &std[i].math);
   }
   // now display
   printf("now displaying the students\n");
  for(i=0;i<5;i++){
      printf("student %d\n", i);
      printf("name:%s student ID:%d sex:%c\n", std[i].name, std[i].id, std[i].sex);
      printf("kor score:%d eng score:%d math score:%d\n",
               std[i].kor, std[i].eng, std[i].math);
   }
}
```