STRUCT & UNION

Table of Contents

- last update : April, 2019

I) STRUCT VS. UNION DATA TYPE

I- a. STRUCT DATA TYPE

Simple abstract data type	struct Color { int r; int g; int b; }; // sizeof(struct Color) == 12
	
	struct Color test;
	test.h = 255; test.w = 255; test.d=255;
	
	// or struct Color test = { 255, 255, 255 };

	typedef struct { int r; int g; int g; } Color;
	
	Color test;
	test.h = 255; test.w = 255; test.d=255;
	// or
	Color test = { 255, 255, 255 };

Nested structure	typedef struct { int base; int height; int depth; Color c ; } Box; Box Test; Test.base = 50; Test.height = 3; Test.depth = 5; Test.c.r = 255, Test.c.g=255; Test.c.b = 0; or ultimately you can also initialize it at the time when it is declared: Box Test = { 50, 3, 5, {255,255,0} };

Array of structure	typedef struct {
	int time;
	float velocity;
	float mass;
	} Object;
	
	void main()
	{
	Object bots[2] = { { 50, 9.5, 10.5 }, { 20, 20, 5.5 } };
	for (int i = 0; i< 2; i++)
	cout << "object-" << i << ": " << bots.velocity / obs.time)* bots.mass << endl;
	}

I- b. UNION DATA TYPE

Like structure, but overlapping memory	typedef union {
	  float fVal; // eg. Located at memory at 0x1234567
	  int iVal; // both iVal and fVal are Located at memory at 0x1234567
	} MyUnionType;
	
	sizeof(myUnionType) == 4 , not 8

II) ADVANCED TOPICS

II- a. ENDIANNESS - DATA ARCHITECTURE

There are two ways regarding the sequential order in which bytes are arranged:
Big endian vs Little Endian

e.g. int A = 0x12345678 and &A = 0x9000
where 0x12 is called highest order byte, and 0x78 is the lowest order byte.
In the memory, the bytes are ordered differently between big vs little endian. See this:

Little Endian		Big Endian	
Address	Contents	Address	Contents
9003	78	9003	12
9002	56	9002	34
9001	34	9001	56
9000	12	9000	78

Try the following:

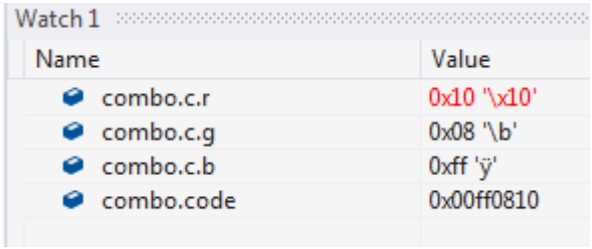
a) Create a short program like the following - to create the union type and initialize the r,g,b only e.g.

```
typedef unsigned char ubyte;
typedef struct { ubyte r; ubyte g; ubyte b; } Color;

typedef union {
      int code;
      Color c;
} ComboType;

ComboType combo;

… in a function, do this :

      combo.c.r = 255; // or 0xff
      combo.c.g = 255; // of 0xff
      combo.c.b = 0;

      printf("%x  %x  %x\n ", combo.c.r, combo.c.g, combo.c.r);
      printf("%x \n",  combo.code);
```

b) Build and run in debugger. Stop right after you initialize the color.
c) Then, also look at the “Watch tab” in the output window.
d) Put the parameter and check out the change in the code field. You should experiment by entering different values.
e) Decide whether your machine uses Big Endian or Little Endian

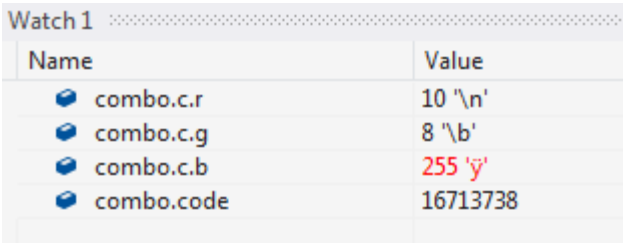
See the following: (one possible architecture)

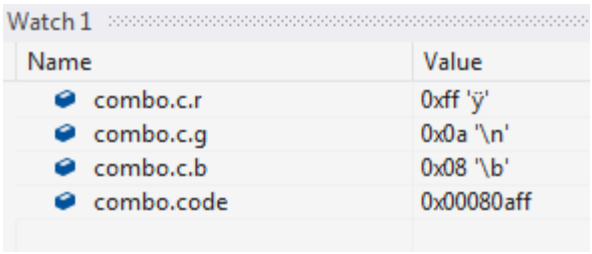
In hex (base-16) display



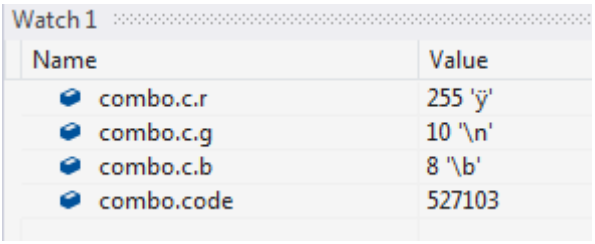
or

In Dec (base-10) display





or



II- b. DATA PADDING

In college, this may fall in computer data architecture or compiler course. Different machine architecture does it slightly different. In order to help the CPU fetch data from memory in an efficient manner, data is being arranged in N-bytes chunk, mostly 4-bytes. This is called data alignment.

Every data type has an alignment associated with it which is mandated by the processor architecture rather than the language itself.

word == 4 bytes for 32-bit processor
word == 8 bytes for 64-bit processor

HOW MEMORY MANAGER ASSIGNS MEMORY SLOTS FOR DATA:

- 1 byte → stored at 1x memory slot
- 2 bytes → stored at 2x memory slot
- 4 bytes → stored at 4x memory slot
- 8 bytes → stored at 8x memory slot

CHECK OUT SOME SAMPLES BELOW:

I highly encourage you to test it out yourself. Observe the addresses for each element through the debugger.

1	typedef struct {
	char c1; // 0
	char c2; // 1
	} TILE;
	sizeof (TILE) == 2

2	typedef struct {
	char c1; // 0
	char c2; // 1
	char ca3; // 2
	} TILE;
	sizeof (TILE) == 3

3	typedef struct {
	char ca; // 0, but 1 is wasted due to data padding
	short ia; // 2-3
	} TILE;
	sizeof (TILE) == 4
	
	memory alignment:
	0 | 1 | 3 | 4
	c1 | - | ia

4

```
struct {
        char c1;        // 0, but 1-3 wasted due to data packing
        int ia;         // 4-7
        char c2;
} TILE;

sizeof ( TILE ) == 12 , not 6
```

memory alignment & padding:

0	1	2	3	4	5	6	7	8	9	10	11
c1	-	-	-	ia				-	-	-	c2

5

```
typedef struct  {
        char c1;        // 0
        char c2;        // 1, but  2-3 wasted
        int ia;         // 5-8
} TILE;
sizeof ( TILE ) == 8 , not 6
```

memory alignment & padding:

0	1	2	3	4	5	6	7
c1	c1	-	-	ia			

6

```
typedef struct {
        int ia;         // 0
        char c1;        // 4
        char c2;        // 5, but 6-7 padded
} TILE;
sizeof ( TILE ) == 8 , not 6
```

memory alignment & padding:

0	1	2	3	4	5	6	7
Ia				c1	x2	-	-

III) BITS FIELD IN STRUCTURE

You can create variables which represent a bit!
(note: ubyte is a user-defined type : typedef unsigned char ubyte)

```
typedef struct {
        ubyte N : 1;
        ubyte NE : 1;
        ubyte E : 1;
        ubyte SE : 1;
        ubyte S : 1;
        ubyte SW : 1;
        ubyte W : 1;
        ubyte NW : 1;
} BitsPACKET;
```

sizeof(BitsPACKET) == 1 byte, not 8 bytes.

WATCH OUT THE BITS ORDER:

```
typedef union  {
        BitsPACKET bits;
        ubyte num;
} unDir;

void main()
{
        unDir dir;
        memset(&dir, 0, sizeof(dir));

        dir.bits.N = 1;
        dir.bits.W = 1;

        // 1  0  0 0 0 0  1 0
        // N  NE E E S SW W NW

        // being stored as  0x41
        // 0  1  0  0  0  0  0   1
        // NW W SW  S  E  E  NE  N

        printf("%d", dir.num);
}
```

IV) RECALLING USING ENUMERATION TYPE

Macros method:	Use Enumeration instead:
#define N 0 #define S 1 #define E 2 #define W 3	enum Directions { N, E, S, W};

e.g. int dir ;
Valid : dir = N;
dir = W;
dir = S;
	e.g. Directions dir;
	Valid : dir = N;
	dir = W ;
	dir = (Directions)3;

Also Valid: dir = 4;	Will not even compile! - dir = 4;

MAY USE IT FOR INDEXING ARRAY

e.g. :
char directions[W];
sizeof(directions) is 3

```
char directions[  ][10] = { "NORTH", "EAST", "SOUTH", "WEST" };
```
sizeof(directions) == 40

DEMONSTRATE FURTHER USAGE:

Sample 1:

```
enum enDir { N, E, S, W,  Last};
int sDir[ Last  ];        // sizeof( sDir ) ==16
```

Sample 2:

```
enum enDir { N, NE, E, SE, S, SW, W, NW, Last};
int sDir[ Last  ];        // sizeof( sDir ) ==32
```

you can even assign sDir[1]= 45 sDir[4] = 180, etc.

Remark: so, you can add any # of elements before the “Last”, your loop will always work without overflow.

V) EXERCISES :

1) What are the actual capacity of the each of the following struct data type.
You should check it out by creating the code. Check it out yourself and watch the address

1	typedef struct {
	int ia1; // 0-3
	char c1; // 4, but 5 padded
	short ia2; // 6-7
	char c2; // 8, but 9-11 padded
	} TILE;
	sizeof (TILE) == ?

2	typedef struct {
	int ia1; // 0-3
	char c1; // 4
	char c2; // 5
	short ia2; // 6-7
	short ia3; // 8-9, but 10-11 padded
	} TILE;
	sizeof (TILE) == ?
	
	Check it out yourself and watch the address

3	typedef struct {
	int ia1; // 0-3
	char c1; // 4
	char c2; // 5
	char c3; // 6
	char c4; // 7
	} TILE;
	sizeof (TILE) == 8

2) Take social security number : ###-##-####

1st 3 numbers : which State
2nd 2 numbers : group
Last 4 numbers : serial number

3) Write a program using Union structure to allow user to enter a full social security number such as "111223456". Then, you serial number without additional expression.

Input Display:
Enter your SSN (#########) : **111223456**

Output:
Region: 111
Group : 22
Serial Number : 3456

4) Create a function to take in a RGB color code, such as 0x66ccff, as stored inside a single variable. Using union structure, you should be able to print out it's individual R,G, B without any extra parsing work.
e.g.
Sample function prototype: void makeColor(unsigned int rgb, ubyte *red, ubyte *green, ubyte *blue)
Your console output should look like this:

Enter the RGB Code : **66ccff**

Your output should look like:

	Hex	Dec
R =	66 or	102
G =	cc or	204
B =	ff or	255

Beware: this sample will read in 66ccff as a hex number, not based-on number.

5) (This exercise will require you to know how to use the "Command Line".) Command line "color" will change the text and background color to a specific color.

e.g. color d5 color d5
color c0 color c0

system("color 6e");

However, your code will ask user to enter *only a single digit* which represents the text color. Then, your code will run the system color command to change both the text & background color where background which is the light version of it.

0 = Black		8 = Gray
1 = Blue		9 = Light Blue
2 = Green		A = Light Green
3 = Aqua		B = Light Aqua
4 = Red		C = Light Red
5 = Purple		D = Light Purple
6 = Yellow		E = Light Yellow

e.g.

mycode 2	This will produce	color 2a
mycode 5	This will produce	color 5d

Restriction: *no conditional expression* is allowed. You may play around with it first : test out how the color sequence changes the console text and background.

6) Write a program to shuffle a deck of cards and display. How efficient your code will be determined by how you create the struct data type.

   Hint: using struct to represent the possible deck of 4 types using Enum.
   Use srand(…) and rand(…) to generate which face and number:
   Face can be: .Club | Spade | Heart | Diamand
   Value can be : 1 to 9 | A | Q | K
   Color : Red | Blue

7) Create a program to print out a 12-months calendar of a given year (use enum type). User should provide the weekday value of Jan 1st, and the year.
   e.g.

```
enum months {  jan, feb, mar,… , dec };
int mdays[ ] = { 31, …. };
```

   When you run your code, your input:

```
myCode   2   2019          // where 2 means it starts from Tue. So, 0 means starting from Sun, etc.
```

   output (should print out all months):

```
Jan, 2019
 Su  M  T  W  Th  F  Sa
                 1  2  3  4  5
 6  7  8  9  10  11  12
 13  14  15  16  17  18  19
 20  21  22  23  24  25  26
 27  28  29  30  31

Feb, 2019
 Su  M  T  W  Th  F  Sa
                    1  2
 3  …

…
Dec, 2019
```

More next page…

8) Take a look of the following permissions bits meaning:

Permission level	3 Permission bits called rwx (i.e. readable | writetable | executable)
Read only	1--
Write only	-1-
Executable only	--1

Permissions sample

Permission bits	
1-1	You can read it and run it, but you cannot over-write to it.
11-	You can read and over-write it, but cannot execute it
111	You can read, over-write and execute it

Write a program to display permission status.

sample input	Output for the sample input
myCode 110	can read | can write | cannot execute
myCode 101	can read | cannot write | can execute
myCode 001 or just 1	cannot read | cannot write | can execute

Do NOT use scanf.

More problem sets will be given only after all these are completed with satisfactory implementation with correct results, of course. This will include:

- File I/O
- nested structure
- pointer to functions.