STRUCT (+ UNION + BITS FIELD)

Table of Contents

I) STRUCT VS. UNION DATA TYPE

A quick review on struct 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. TIME STRUCTURE

Instinct time structure:

```
struct tm
{
    int tm_sec;   // seconds after the minute - [0, 60] including leap second
    int tm_min;   // minutes after the hour - [0, 59]
    int tm_hour;  // hours since midnight - [0, 23]
    int tm_mday;  // day of the month - [1, 31]
    int tm_mon;   // months since January - [0, 11]
    int tm_year;  // years since 1900
    int tm_wday;  // days since Sunday - [0, 6]
    int tm_yday;  // days since January 1 - [0, 365]
    int tm_isdst; // daylight savings time flag
};
```

e.g.

```
time_t now;
struct tm *timeinfo;

time(&now);
timeinfo = localtime(&now);
printf("Current local Date and Time: %s\n", asctime(timeinfo));
printf("[%d %d %d %d:%d:%d]\n",
        timeinfo->tm_mon + 1, timeinfo->tm_mday,
        timeinfo->tm_year + 1900,
        timeinfo->tm_hour, timeinfo->tm_min, timeinfo->tm_sec);
```

Output :

```
Current local Date and Time: Wed Jul 17 10:30:45 2019

[7 17 2019 10:30:45]
```

II) UNION DATA TYPE

Review the Union section chapter 16 in the book.

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

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) ADVANCED TOPICS

V- 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
9000	78	9000	12
9001	56	9001	34
9002	34	9002	56
9003	12	9003	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.

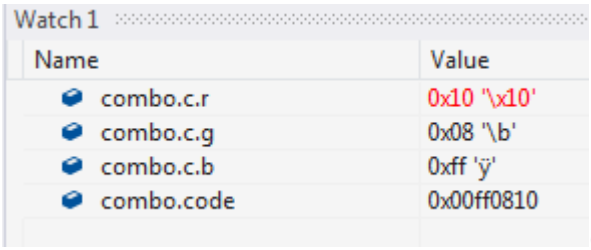
```
typedef unsigned char ubyte;
struct Color { ubyte r; ubyte g; ubyte b; };

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.

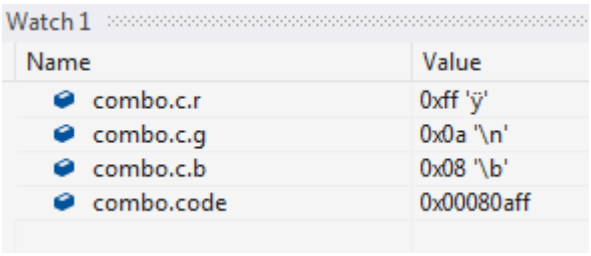
d) Put the parameter and check out the change in the code field. You should experiment by entering different values.
e) Decide whether your machine uses Big Endian or Little Endian

See the following: (one possible architecture)

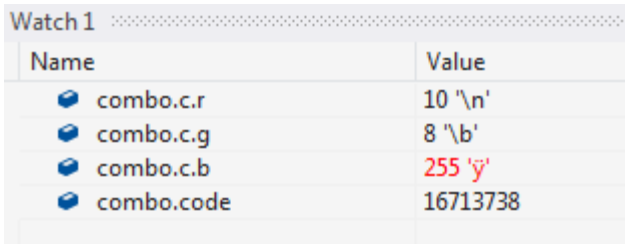
In hex (base-16) display



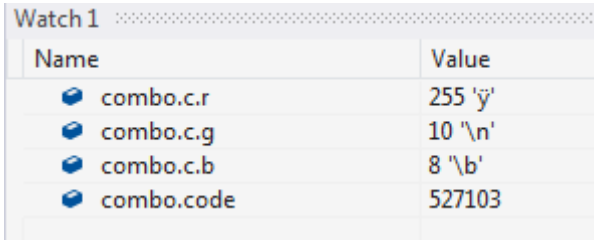
or



In Dec (base-10) display



or



V- 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.

<table>
<tr><td>1</td><td>typedef struct {
char c1; // 0
char c2; // 1
} TILE;
sizeof (TILE) == 2</td></tr>
<tr><td>2</td><td>typedef struct {
char c1; // 0
char c2; // 1
char ca3; // 2
} TILE;
sizeof (TILE) == 3</td></tr>
<tr><td>3</td><td>typedef struct {
char ca; // 0, but 1 is wasted due to data padding
short ia; // 2-3
} TILE;
sizeof (TILE) == 4

memory alignment:
<table><tr><td>0</td><td>1</td><td>3</td><td>4</td></tr><tr><td>c1</td><td>-</td><td colspan="2">ia</td></tr></table></td></tr>
<tr><td>4</td><td>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:
<table><tr><td>0</td><td>1</td><td>2</td><td>3</td><td>4</td><td>5</td><td>6</td><td>7</td><td>8</td><td>9</td><td>10</td><td>11</td></tr><tr><td>c1</td><td>-</td><td>-</td><td>-</td><td colspan="4">ia</td><td>-</td><td>-</td><td>-</td><td>c2</td></tr></table></td></tr>
<tr><td>5</td><td>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:
<table><tr><td>0</td><td>1</td><td>2</td><td>3</td><td>4</td><td>5</td><td>6</td><td>7</td></tr><tr><td>c1</td><td>c1</td><td>-</td><td>-</td><td colspan="4">ia</td></tr></table></td></tr>
</table>

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	-	-

V- c. 3.B) MORE ON MEMORY ALIGNMENT

(NOTE: not in struct)

Example for my current Processor - Intel Core i7-7500U, it needs to consider two things – P & A:
where P = the size of a pointer (based on the CPU architecture)
A = the alignment required (a Word), expressed in 2^x bytes.

SAMPLES OF SIMPLE VARIABLES DECLARATION

e.g. 1:

```
char  c1;
int i1;
char c2;
int i2
```

e.g. 2:

```
int i1;
int i2;
char c1;
char c2;
```

Word	Word	Word			
1… 4	5 … 8	9	10	11	12
i1	i2			C1	C2

e.g. 3:

```
char  c1;
short i1;
char c2;
short i2;
```

Word				Word				Word			
1	2	3	4	5	6	7	8	9	10	11	12
I1				I2						C1	C2

e.g. 4:

```
struct1 s1; // 7bytes structure
struct12 s2; // 3bytes structure
short i1,
short i2;
struc13 s2; // 12bytes structure
```

Word	Word	word	Word	Word	Word	
1… 4	5 … 8	9. .12	13… 16	17… 20	21- 24	25- 28
S1			S2		I1	I2

VI) 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) Remember the digital display exercises you have done way back in chapter 8? Write a program to display the current time like the digital segments display simulation. Your program should constantly update time until you hit 'C' as cancel.

   Hint :
   - Use system("clear") to refresh the display. You will need to include a special header file – refer to the book.

3) Write a program using Union structure to allow user to enter a full social security number such as "111-22-3456". Then, you serial number without additional expression.

Input Display:
Enter your SSN (###-##-####) : **111-22-3456**

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`

to do this in your program, you need to call:

```
system("color 6e");
```

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

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.
e.g.

Sample commands	This will produce

mycode 2	`color 2a`
mycode 5	`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) (This exercise will require you to know how to do basic file I/O.) Write a program to change the color and background color of text inside a pre-made html file. You may use the sample in the text box below.

Steps to follow:
Step 1: Create a html file like below using a text editor like notepad or notepad++, and modify the color and background-color. Make sure you know where you save it.

In this sample, it is saved in **c:/users/you/Documents/test.html**

```
<html>
<body style="background-color:#00ffff;color:#ff0000">
<div >
This is a test1.<br/>
This is a test2.<br/>
</div>
</body>
</html
```

To test this: Access the file at your browser. Type this in your URL:
file:///c:/users/you/documents/test.html
Your browser page may look something like this:

Step 2: Write a program to ask user to input the value of Red, Green, and Blue. Then, it should produce the final color value in Hexadecimals.

Sample console input: (red bold font indicates user input)

```
Enter R: (0 <=x <= 255) :  255
Enter G: (0 <=x <= 255) :  0
Enter B: (0 <=x <= 255) :  255
```

Step 3: Your program should open the html file, read content into the local memory buffer, modify the color and background-color fields, save it back out to the same file.

Step 4: Access it via a browser to check the changes.

==========
Some background information:

Color Code usually presented in hex, $RRGGBB_{16}$.

For example: $66CCFF_{16}$, i.e. R== 66_{16}, G=0xCC_{16}, B=FF_{16}, that gives cyan.
Sample function prototype:

```
int  createColor( unsigned char red, unsigned char green, unsigned char blue) ;
        // may create your own data type ubyte instead of unsigned char
```

If you want to validate the color, you can check this out :
http://www.w3schools.com/colors/colors_picker.asp

---

7) Write a program to:
   a. shuffle a deck of cards.
   b. Distribute them to 4 players.
   c. Display what these 4 players have.

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 | Diamond
Value can be : 1 to 9 | A | Q | K
Color : Red | Blue

How efficient your code will be determined by how you create the struct data type.

---

8) 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
…

For example, utilize enumeration for moths
enum months { 31, 28, 31, …. , 30, 31}; // days for each month
char mNames[12] = { “January”, … “December” };

---

9) Write a program to display permission status by utilizing “bits field in structure”.

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

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

Do NOT use scanf.

10) Write a program to :

a. Read a pre-existing csv file (delimited by ‘,’)
b. Sort it based on input from command line.
c. Your code must be able to sort by one of the following 3 data types:
    i. string
    ii. Int
    iii. date (in mm/dd/yyyy format)
d. Write the sorted data back out to the csv file.
e. Open it with excel (or google sheet) to view the sorted data.

Assumption:

- May allow maximum 255 columns of data.
- Each column may contain max 255 characters.

e.g. Spreadsheet: just a simple sample. You should always try with very simple sample like below, then add in more columns, values, etc. for testing.

Fname	Lname	Age	DOB
James	Neil	15	01/01/2000
Amy	Fu Too	10	10/09/2005
Jared	Py Si	9	03/10/2006

Csv file before sorting:

```
Fname, Lname, Age, DOB
James, Neil, 15, 01/01/2000
Amy, Fu Too, 10, 10/09/2005
Jared, Py Si, 9, 03/10/2006
```

e.g. Your executable name is : parseThis.exe

Command line:

```
parseThis  <field name>  <A | D >
```

So:

```
parseThis    Age    A
```

(This means sort the data by Age in Ascending order. D == descending)

Csv file after sorting:

```
Fname, Lname, Age, DOB
Jared, Py Si, 9, 03/10/2006
Amy, Fu Too, 10, 10/09/2005
James, Neil, 15, 01/01/2000
```

HINT: Require you to use:

- struct and union

- array of pointers to functions
- enum
- string manipulation such as parsing
- command line arguments.

Efficiency of your code will depend on how you design your structure.