STRUCT & UNION

TABLE OF CONTENTS

1) *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 };
	
	OR
	
	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 };

	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} };

Structure Array	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;
	}

2) UNION DATA TYPE

```
typedef union {
   float fVal;
   int iVal;
} MyUnionType;
```

sizeof(myUnionType) == 4 , not 8

Try the following:

```
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("%d", combo.code);
```

Try the following:
a) Create a short program to create the union type and initialize the r,g,b only.
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.

In hex (base-16) display

Watch 1

Name	Value
combo.c.r	0x10 '\x10'
combo.c.g	0x08 '\b'
combo.c.b	0xff 'ÿ'
combo.code	0x00ff0810

or

In Dec (base-10) display

Watch 1

Name	Value
combo.c.r	10 '\n'
combo.c.g	8 '\b'
combo.c.b	255 'ÿ'
combo.code	16713738

Watch 1

Name	Value
combo.c.r	0xff 'ÿ'
combo.c.g	0x0a '\n'
combo.c.b	0x08 '\b'
combo.code	0x00080aff

or

Watch 1

Name	Value
combo.c.r	255 'ÿ'
combo.c.g	10 '\n'
combo.c.b	8 '\b'
combo.code	527103

3) ADVANCED TOPICS - DATA ALIGNMENT & PADDDING

3.A) DATA STRUCTURE ALIGNMENT WITH 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:
I highly encourage you to test it out yourself. Observe the addresses for each element through the debugger.

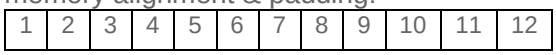
	typedef struct {
	  char c1; // 0
	  char c2; // 1
	} TILE;
	sizeof (TILE) == 2

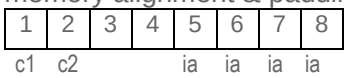
	typedef struct {
	  char c1; // 0
	  char c2; // 1
	  char ca3; // 2
	} TILE;
	sizeof (TILE) == 3

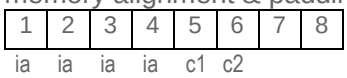
	typedef struct {
	  char ca; // 0, but 1 is wasted due to data padding
	  short ia; // 2-3
	} TILE;
	sizeof (TILE) == 4
	
	memory alignment:
	
	1 2 3 4
	c1 ia ia

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

<table>
<tr><td></td><td>sizeof (TILE) == 12 , not 6

memory alignment & padding:
 
</td></tr>
<tr><td></td><td>typedef struct {
char c1; // 0
char c2; // 1, but 3-4 wasted
int ia; // 4-7
} TILE;
sizeof (TILE) == 8 , not 6

memory alignment & padding:
 
</td></tr>
<tr><td></td><td>typedef struct {
int ia; // 0
char c1; // 4
char c2; // 5, but 6-7 padded
} TILE;
sizeof (TILE) == 8 , not 6

memory alignment & padding:
 
</td></tr>
<tr><td></td><td>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) == 12 , not 6</td></tr>
<tr><td>2</td><td>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) == 12 , not 10</td></tr>
</table>

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

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 11	10 12		
i1	i2			1	2

e.g. 3:

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

Word				Word				Word			
									0	1	2
1				2						1	2

e.g. 4:

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

<table>
<tr><td>Word</td><td>Word</td><td>word</td><td>Word</td><td>Word</td><td>Word</td><td></td></tr>
<tr><td>1…4</td><td>5 …8</td><td>9. .12</td><td>13 … 16</td><td>17 … 20</td><td>21-24</td><td>25-28</td></tr>
<tr><td colspan="4">S1</td><td colspan="3">S2</td><td></td><td>1</td><td></td><td></td><td>2</td><td></td><td></td></tr>
</table>

4) BITS FIELD IN C 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);
       }
```

5) Recalling using Enumeration Type

Macros method:	Use Enumeration instead:

#define N 0
#define S 1
#define E 2
#define W 3
 	
e.g. int dir ;
Valid : dir = N;
dir = W;
dir = S;
 	
Also Valid: dir = 4;
	enum Directions { N, E, S, W};
	
	e.g. Directions dir;
	Valid : dir = N;
	dir = W ;
	dir = (Directions)3;
	
	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.

6) EXERCISES :

Take social security number : ###-##-####

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

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

1) Create a function to take in a RGB color code, such as FFFF00, then it should generate the individual R. G, B value. 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:
R (in hex) = 66 or 102
G (in hex) = cc or 204
B (in hex) = ff or 255

2) Command line "color" will change the text and background color to a specific color.

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

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
7 = White	F = Bright White

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.

myCode 2 `color 2a`
myCode 5 `color 5d`

One restriction, you *no conditional expression* is allowed. You may play around with it first : test out how the color sequence changes the console text and background
e.g. system("color 6e");

3) Write a program to:
Change the color and background color of text inside a pre-made html file.

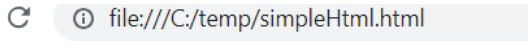
Access the html file from a browser to verify your change.

The background-color should be simply compliment of the color bits, i.e. if bit ==1 , change it to 0. Access the html file from browser to check your change.

Step 1:

Create a html file like below, and modify the color and background-color.

```
<!—Create this html file. Just call it simple.html .  Then -->
<html>
<body>
<p style="background-color:#00ff00;color:#ff00ff">This is a test</p>
</body>
</html
```

To test this: Access the file at your browser. Eg. 

Step 2:

Write a function 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:

Read in the html file, modify the color and background-color, save it.

Access it via a browser to check your work.

==========

Some background information:

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

For example: 66CCFF$_{16}$, i.e. R== 66$_{16}$, G=CC$_{16}$, B=FF$_{16}$, that gives cyan.

Sample function prototype:

```
int  createColor( unsigned char red, unsigned char green, unsigned char blue) ;
        // 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

4) Refer to the Bits Fields in Structure sample above. You will write two functions:

```
void setBits(unDir *, enDir)
bool getBits(unDir, enDir)
```

If you do not know pointer, just make the unDir field as a global instead. Assuming a global field unDir Direction already exists. Your setBits(…) function will set the bit field inside this structure. Thus, your 1st function prototype will be :

```
void setBits(enDir)
bool getBits(enDir)
```

5) 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 3 data types:
      i. String | Int | 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.

e.g.
spreadsheet:

Fname	Lname	Age	DOB
James	Neil	15	1/1/2000
Amy	Fu Too	10	10/9/2005
Uma	Py Si	9	3/10/2006

Csv file before sorting:

Fname, Lname, Age, DOB
James, Neil, 15, 01/01/2000
Amy, Fu Too, 10, 10/09/2005
Uma, 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
Uma, Py Si, 9, 3/10/2006
Amy, Fu Too, 10, 10/9/2005
James, Neil, 15, 1/1/2000

HINT: Use struct and union
Efficiency of your code will depend on how you design your structure.

OPTIONAL:

6) Write a program to allow two users to simulate a partial Chess game with only 2 colored (red, black) groups of 8 pieces/group:

1 Queen, 1King, and 2 Bishops, 4 Pawns

Startup:

	B	Q	K	B	
	P	P	P	P	
	P	P	P	P	
	B	Q	K	B	

As you see this is not a complete 8x8 chess board but only 6x6, nor does it have all pieces. The goal of this program is not to do AI Chess game, but to exercise proper design with struct, enum, functions, and possibly union if applicable.

You do not need to be a proficient chess player, but need to know the rules of a king, queen and bishop, and pawn.

Legal moves:

- The bishop: only in a straight line diagonally for any number of squares
- The queen: in any number of squares in a straight line horizontally, vertically or diagonally.
- The king: in any direction, including diagonally, but it can only move one square at a time.
- The pawn:
  - Move one square forward only, not backward.
  - If it is the pawn's first move, it can move one or two squares directly forward; but only one at a time after 1st move.

To capture:

- The bishop, queen and king:
  - capture others in their legal path, except when there is another piece (like its own color) in the way.
- The pawn:
  - cannot capture a piece directly in front of it.
  - Capture a piece only by moving one square forward diagonally.