Lecture 5

1. Constants

`const` is a type-qualifier that defines the accessibility of the variable. The code

```
int main(void)
{
    const int x = 5;
    x = 0;
}
```

will raise an error

```
assignment of a read-only variable x
```

as we can't change the value of the `const` variable. However, the code

```
int main(void)
{
    const int x = 5;
    int *p;
    p = &x;
    *p = 0;
}
```

will change the value of `x` and won't raise an error.

So, the `const` keyword change the writeability to be prohibited throught the variable name but can be changed by a pointer points to the address of that variable if the pointer points to a variable without the `const` keyword.

In other words,

```
int x = 3;       // the code will behave the same if it was const int x= 3;
int *p1;        // points to an int
const int *p2; // points to a const int
p1 = &x;
p2 = &x;
*p1 = 2; // this will work fine, will give a warning
*p2 = 5; // will raise an error
```

will raise an error

```
error: assignment of read-only location '*p2'
 *p2 = 5;
```

The `const int *p2` can be useful for a variable that is not `const` if we want to pass the variable by reference to save copying time and new variable memory and at the same time make sure that the value of the variable doesn't change in the function through this pointer.

```
void display(const int *p)
{
    int y;
    //By Mistake :(
    *p = y; // should be y = *p;
}
```

will raise an error

```
assignment of read-only location '*p'
```

code	variable value	p value	Example
`int *p;`	changable	changable	pass by refernce
`const int *p;`	const	changable	pass by reference of a variable whose value to be not changed
`int * const p;`	changable	const	memory and registered management
`const int * const p;`	const	const	status registers

2. Integer Constant

- Decimal format of normal numbers. 1, 1000, 5000
  - Default type: int, long
- Hex format 0x00, 0X1a, 0xAA
  - Default type: unsigned int, unsigned long
- Octal format 012, 044
  - unsigned int, unsigned long
- Binary format.

3. Enum Constant

You can give names to numbers for better readability. Enum values are by default signed integral constant.

```
typedef enum{
    RED,
    GREEN
} ColorType;

int main(void)
{
    ColorType color;
    color = RED;
    printf("%d ", color);
    color = GREEN;
    printf("%d ", color);
}
```

will result in

```
0 1
```

As for enum if the first value isn't declared, then it becomes 0. If the next name is not declared then it becomes the last value + 1.

```
typedef enum{
    BLUE,
    RED = 10,
    GREEN
} ColorType;

int main(void)
{
    ColorType color;
    color = BLUE;
    printf("%d ", color);
    color = RED;
    printf("%d ", color);
    color = GREEN;
    printf("%d ", color);
}
```

will result in

```
0 10 11
```

As `BLUE` isn't declared so it is 0, `RED` is declared with 10, `GREEN` isn't declared so its value become the last value `RED` which is 10 + 1 so it is 11.

4. Real Constants

- The default data type of the floating number is `double`.
- Real constants suffixed by f or F are float. 1.0f
- Real constants suffixed by l or L are long double. -4.2e-3L

5. Char Constants

will be written later

6. Voltaile

- used when telling the compiler to not discard that variable or an operation related to that variable for any reason, mostly that it is not used or different dereferncing of a hardware register.
- Useful with the variables of the interrupts and the addresses of the registers.

7. Preprocessings and Pragmas

- Preprocessor: a tool that runs before compilation to prepare the code before being compiled speciallty with text replacement.
- All preprocessors start with `#`.
- Macros are preprocessors that can be symbolic macros, function-like macros or conditional compilation.

Symbolic macros:

```
#define PI 3.14
```

Function-like macros:

```
#define MUL(X, Y) X*Y
```

Conditional compilation

```
 #define COND 0

#if COND == 0
// this will be compiled
int main()
{
    printf("Hi");
}

#elif COND == 1
// this will not be compiled or replaced
int main()
{
    printf("Bye");
}

#endif
```