Side effects

The actual order in which expressions are evaluated is not specified for most of the operators in C. Because this sequence of evaluation is determined within the compiler depending on context, some unexpected results may occur when using certain operators. These unexpected results are caused by side effects.

Function calls, nested assignment instructions and increment and decrement operators may cause side effects.

Examples:

```
x = f( ) + g( );  /*Undefined calling order */
printf("%d %f\n", ++n, pow(2,n));
a[ i ] = i++;     /* The increment of i may occur before or
                     after the subscript is evaluated. */
```

The *while* statement

while (expression) statement

The **while** statement evaluates a control expression before each execution of the loop body. If the control expression is true (nonzero), the loop body is executed.

Example:

```
while ((c = getchar( )) == ‘ ‘ || c == ‘\n’ || c == ‘\t’)
   ; /*empty statement – omit white characters */
```

The *while* and the *for* loops

The for statement:

```
   for (expression1; expression2; expression3)
         statement
```

is equivalent to the following code:

```
expression1;
while (expression2)
{
   statement
   expression3;
}
```

Example *1*

```
# include <stdio.h>
/* program prints the following numbers from
   1 to 100 and their squares on the screen */

int main (void)
{
int i;
   for (i =1; i <=100; i++)
               printf("Number %d\t%d\n", i, i*i);
return 0;
}
```

Example *2*

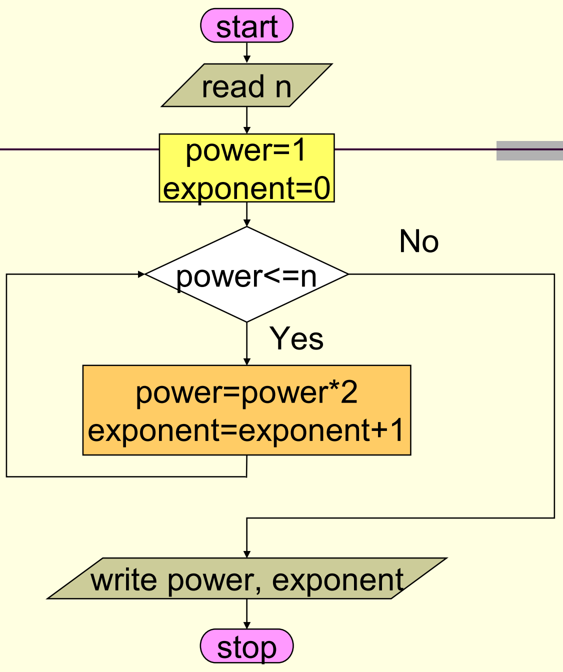
```
# include <stdio.h>
/* program prints the following numbers from
   1 to 100 and their squares on the screen */
void main ()
{
int i =1;
   while (i <=100)
   {
        printf("Number %d\t%d\n", i, i*i);
         i++;
   }
}
```

Example 13

Problem: For a given natural number ***n*** compute the smallest power of two greater than ***n***.

Discussion:
Input: n
Output: power of two, exponent
Check the increasing powers of two in a loop, while the condition ***power*** <= ***n*** is true.

Algorithm:
- read ***n***;
- set initial values (*exponent:=0; power:=1*;);
- check, if ***power*** <= ***n***;
- while ***power*** <= ***n***, multiply it by 2 and increment ***exponent***;
- write the results.



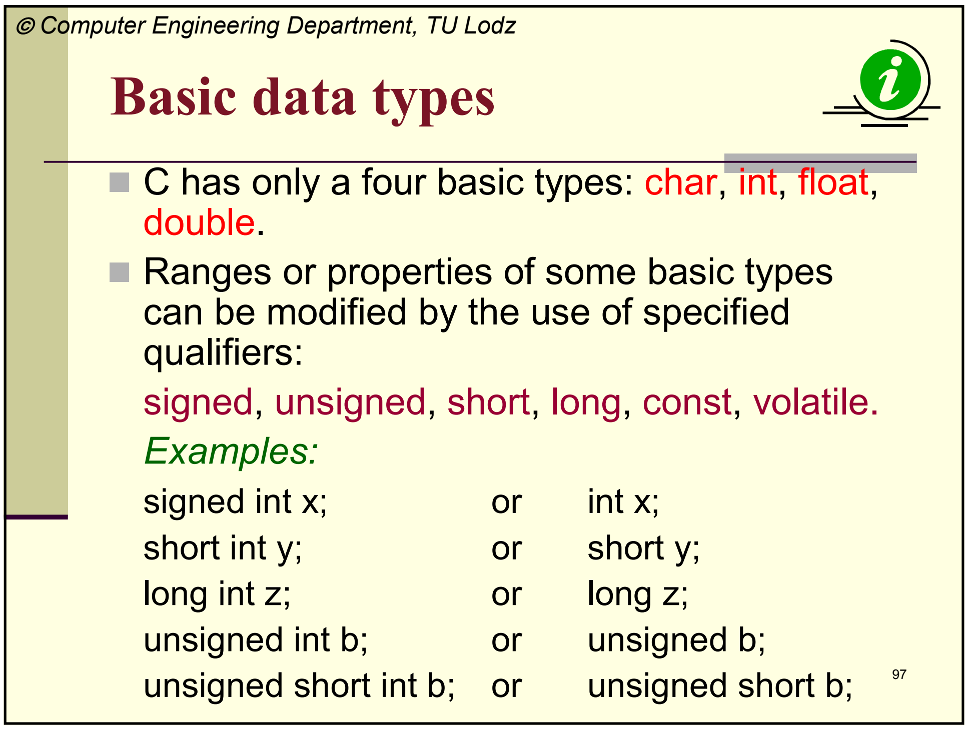
```c
/*File Power_two.c
                Program finds the smallest power of two greater than n*/
#include <stdio.h>
int main()
{
   int, n, power, exponent;
   printf("Program finds the smallest power of two\n");
   printf("greater than the given n\n");
   printf("Enter natural number n: ");
   scanf("%d", &n);
   exponent =0;    power =1;     // initial values
   while(power<=n)               // loop condition
   {
     power = power *2;
     exponent = exponent +1;
   }                             // end of the loop
  printf("The smallest power of 2 greater than %d  is %d for the exponent
     %d\n", n, power , exponent);
  getchar();
  return 0;
}
```

Standard mathematical functions
<math.h> header file

sin(x) -sine x (argument in radian),
cos(x)-cosine (argument in radian),
tan(x) - tangent x (argument in radian),
asin(x) - arc sine x (result in the range [$-\pi/2$, $\pi/2$]),
acos(x) - arc cosine x (result in the range [0 , π]),
atan(x) – arc tangent x (result in the range [$-\pi/2$, $\pi/2$]),
atan2(x,y) – arc tangent x/y (result in the range [$-\pi$, π]),
sinh(x) - hyperbolic sine of x,
cosh(x) - hyperbolic cosine of x,
tanh(x) - hyperbolic tangent of x,

Standard mathematical functions
<math.h> header file

exp(x)- exponential function of x: e^x,
log(x) - the natural logarithm of *x*: ln(x), x > 0,
log10(x) - the base-ten logarithm of *x*: $\log_{10}(x)$, x > 0,
pow(x,y) – power function: x^y, a domain error occurs if:
x=0 and y≤0, or x<0 and y is not an integral value,
sqrt(x) - the nonnegative square root of *x*: $\sqrt{x}, where: x \geq 0$
fabs(x) - the absolute value of a floating-point number *x:* |x|
ceil(x)- the smallest integral value not less than x,
floor(x) - the largest integral value not greater than *x,*
result of double type.



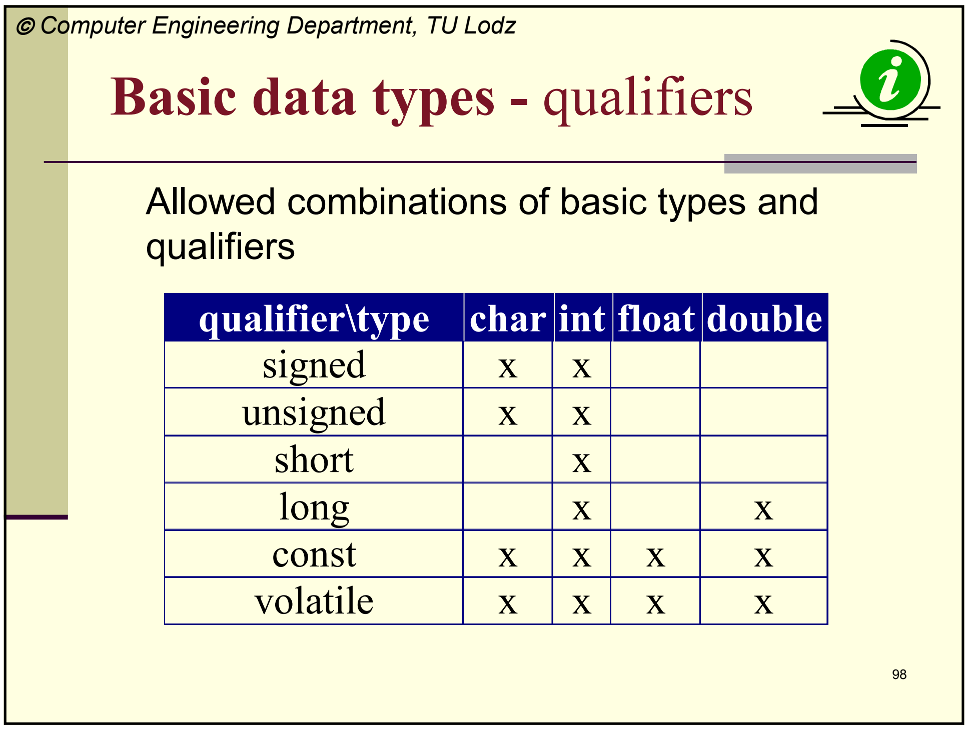
Basic data types

- C has only a four basic types: char, int, float, double.
- Ranges or properties of some basic types can be modified by the use of specified qualifiers:

  signed, unsigned, short, long, const, volatile.

  Examples:

signed int x;	or	int x;
short int y;	or	short y;
long int z;	or	long z;
unsigned int b;	or	unsigned b;
unsigned short int b;	or	unsigned short b;



Basic data types - qualifiers

Allowed combinations of basic types and qualifiers

qualifier\type	char	int	float	double
signed	x	x		
unsigned	x	x		
short		x		
long		x		x
const	x	x	x	x
volatile	x	x	x	x

Sizes and Ranges of Data Types for 32-bits (16-bits) machines

Type	Range	Size in bytes
char	-128..127	1
unsigned char	0..255	1
short	-32768..32767	2
unsigned short	0..65535	2
int	$-2^{31} \ldots 2^{31}-1$ $(-2^{15} \ldots 2^{15}-1)$	4 (2)
unsigned	$0 \ldots 2^{32}-1$ $(0 \ldots 2^{16}-1)$	4 (2)
long	$-2^{31} \ldots 2^{31}-1$	4
unsigned long	$0 \ldots 2^{32}-1$	4
float	$3.403*10^{38} \ldots 1.175*10^{-38}$	4
double	$1.798*10^{308} \ldots 2.225*10^{-308}$	8

const and volatile qualifiers

- const and volatile qualifiers may be used in declarations before any type.
- The const type qualifier is used to qualify an object whose value cannot be changed – declaration of constant.
- The volatile storage class is specified for those variables that can be modified in ways unknown to the compiler. Thus, if an object is declared volatile, every reference to the object in the source code results in a reference to memory in the object code.

Examples

```
const float pi = 3.14; // definition and initialization

volatile int temperature;
volatile float pressure;
printf("Carrent temperature = %d oC, and pressure=
        %f Pa\n", temperature, pressure);
```

Symbolic constants, #define directive

The #define directive has the following syntax:

#define ***identifier*** *replacement-list newline*

The replacement list is a sequence of preprocessing tokens, which is substituted for every occurrence of that macro identifier in the program text.

Examples:

```
#define PI  3.14
#define MACHINES_NUMBER 4
#define ENGINES_NUMBER 4
```