C Programming Language

First C Program: Hello World!

```
Line 1:     /*
Line 2:         code: "hello world!“
Line 3:         author: Jack
Line 4:     */
Line 5:     #include <stdio.h>
Line 6:     int main()  // this is head of function
Line 7:     {
Line 8:         // this is a comment again
Line 9:         printf("Hello, world!\n");
Line 10:        return 0;
Line 11:    }
```

Four Basic Data Types

- In C, there are four basic data types:

  1. char 2. int 3. float 4. double

- Use data type to define Variables:

```
char   c = 'A';
int    age = 4;
float  weight = 142.5;
```

- Keywords and identifiers:

  1. Keywords are reserved names, they are all in lower case
  2. Identifiers are user defined variable or function names, formed by letters/digits/underscore _ they can not start with a digit

int printf(format string, arg2, …)

- It is to print message or result on the output device (i.e. monitor). It is defined in stdio.h header file.
- Returns the number of characters it prints out.
- Format string is always needed, the rest of arguments should match format specifiers.

Example:

```
printf( “You are in class %c ”,      c );
printf(“Age =%d, weight  is %f \n”, age, weight);
printf(“Weight per age is %f \n”, weight/age);
```

Format specifiers

- There are several format specifiers-The one you use should depend on the type of the variable you wish to print out. The common ones are as follows:

Format Specifier	Type
%d	int
%c	char
%f	float
%lf	double
%s	string

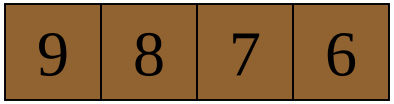
To display a percentage sign by %%

Forward slash \ is called escape character for special use:

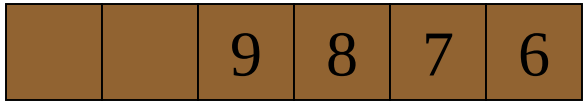
\n \t \\ \”

Example

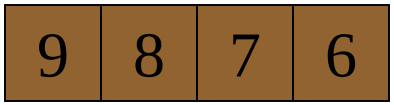
- printf(“%d”,9876)



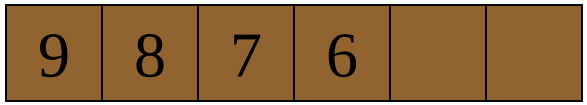
- printf(“%6d”,9876)



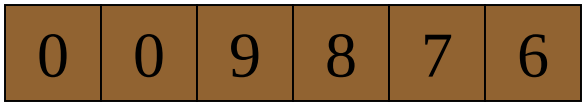
- printf(“%2d”,9876)



- printf(“%-6d”,9876)



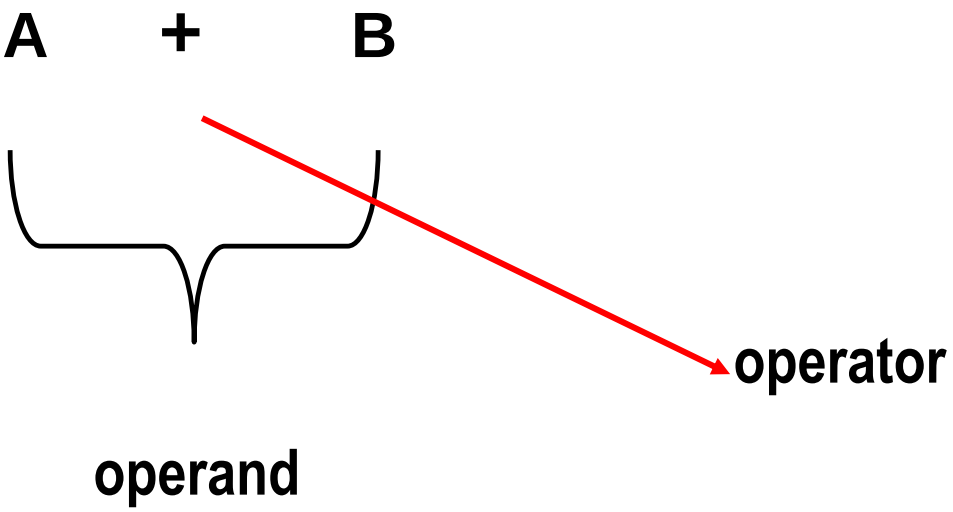
- printf(“%06d”,9876)



Expression

- An expression is a combination of operands (constants, variables, numbers) connected by operators and parenthesis.

- Example :



Types of expression

- Arithmetic expression : An expression that involves arithmetic operators: binary operator + - * / %

  unary operator ++ --

- Relational or Logical expression : An expression that involves relational (>, <, >=, <=, ==, !=) or logical (&&, ||, !) operators.

- C language doesn't provide a logical data type.

  Result of Relational or Logical expression is an integer: **1** (means TRUE) or **0** (means FALSE)

Assignment operator =

- It is used to **assign** values to variables:
  variable_name = expression;

- Multiple assignment:
  int j=k=m=0;

- Compound assignment:

  j= j+10; can be written as j **+ = 10;**

  similarly,

  m - = 100; n *= 5; x /= 10.5; i %= 3; all valid

Increment operator (++)

- The increment (++) operator adds **1** to its operand. It is an unary operator.

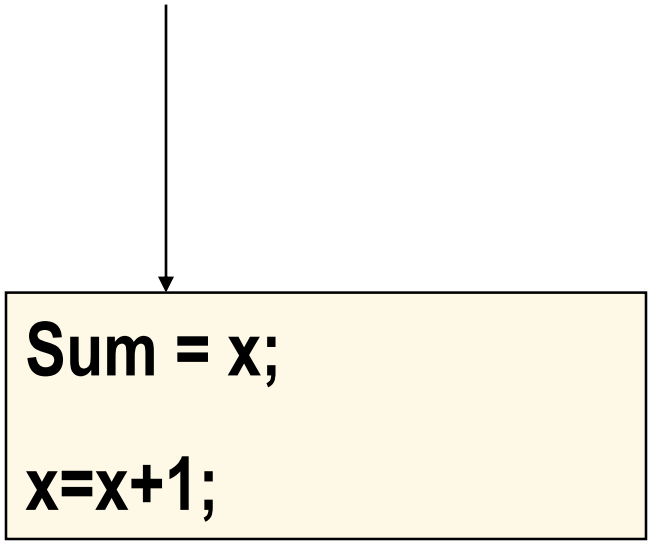
  n = n +1; can be replaced by **++ n; or n ++;**

- Postfix Increment **(n ++)** : It increments the value of n after its value is used.

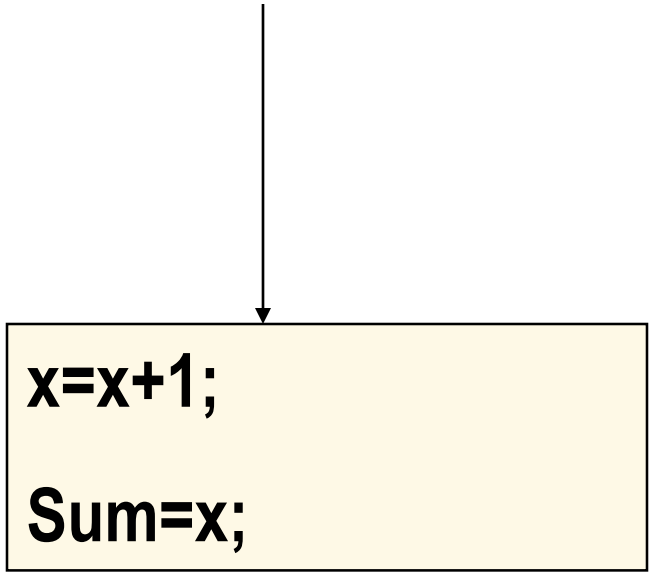
- Prefix Increment **(++ n)** : It increments the value of n before it is used.

Example

sum=x++;



sum = ++x;



sum=x=x+1;

Decrement operator (--)

- The decrement (--) operator subtracts **1** from its operand.

  j = j - 1; can be replaced by **-- j; or j --;**

- Postfix decrement (j --) : In this case value of operand is fetched before subtracting 1 from it.

- Prefix decrement (-- j) : In this case value of operand is fetched after subtracting 1 from it.

Relational operators

A relational operator is used to compare two values and the result of such operation is always an integer:

1 (means TRUE) or **0** (means FALSE)

<	less than	x < y
>	greater than	x > y
<=	less than or equal to	x <= y
>=	greater than or equal to	x >= y
==	is equal to	x = = y
!=	is not equal to	x != y

Logical operator

&&	Logical AND	x && y
||	Logical OR	x || y
!	Logical NOT	!x

- The operand of a logical operator can be of any kind: arithmetic, relational, or logical.

  For arithmetic expression as operand, any non-zero value means TRUE while zero means FALSE:

- The result of a logical expression is always an integer:

  Example: (-0.1)&&(4+5) → 1 (means TRUE)

  (0) || (7<=5) → 0 (means FALSE)

  !(5.5!=4) → 0 (means FALSE)

Flow control

- Loop iterations:

```
for( i=1; i<=9; i++ ) {
        statements;
}
```

- Condition:

```
if( any expression ) {
        statements;
}
else if {
        statements;
}
else { …}
```

9 X 9 Multiplication Table

```
0  1  2  3  4  5  6  7  8  9
1  1  2  3  4  5  6  7  8  9
2  2  4  6  8 10 12 14 16 18
3  3  6  9 12 15 18 21 24 27
4  4  8 12 16 20 24 28 32 36
5  5 10 15 20 25 30 35 40 45
6  6 12 18 24 30 36 42 48 54
7  7 14 21 28 35 42 49 56 63
8  8 16 24 32 40 48 56 64 72
9  9 18 27 36 45 54 63 72 81
```

```
Line 1:   #include <stdio.h>
Line 2:   int main()
Line 3:   {
Line 4:       int i,j;
Line 5:       for( j=0; j<=9; j++ ) {
Line 6:           printf("%3d", j);
Line 7:       }
Line 8:       printf("\n");
Line 9:       for( i=1; i<=9; i++ ) {
Line 10:          printf("%3d",i);
Line 11:          for( j=1; j<=9; j++ ) {
Line 12:              printf("%3d", i*j);
Line 13:          }
Line 14:          printf("\n");
Line 15:      }
Line 16:  }
```

int scanf(format string, arg2, …)

- It is to read in data from a input device (i.e. keyboard). It is defined in stdio.h header file.
- Returns the number of items it successfully reads in.
- Arguments other than the format string are addresses of variables.

Example:

```
printf(“ Please enter your age and weight: ”);
scanf(“ %d, %f ”, &age, &weight);
```

scanf( )

- **The following two codes are doing the same:**

```
#include<stdio.h>
int main()
{
int n;
printf("enter the value");
printf("%d", scanf("%d",&n) );
return 0;
}
```

```
#include<stdio.h>
int main()
{
int n, nread;
printf("enter the value");
nread= scanf("%d",&n);
printf("%d", nread);
return 0;
}
```

n X n Multiplication Table

```
Line 1:   #include <stdio.h>
Line 2:   int ntable(int );
Line 3:   int main() {
Line 4:         int n;  printf(“enter n: “);
Line 5:         scanf(“%d”, &n);   ntable(n);
Line 6:   }
Line 7:   int ntable(int n)
Line 8:   {
Line 9:         int i, j;
Line 10:        for( j=0; j<=n; j++ ) {
Line 11:              printf("%3d", j);
Line 12:        }
Line 13:        printf("\n");
Line 14:        for( i=1; i<=n; i++ ) {
Line 15:              printf("%3d",i);
Line 16:              for( j=1; j<=n; j++ ) {
Line 17:                    printf("%3d", i*j);
Line 18:              }
Line 19:              printf("\n");
Line 20:        }
Line 21   }
```

```
#include <stdio.h>
int main()
{
      int i, j;
      for( j=0; j<=9; j++ ) {
            printf("%3d", j);
      }
      printf("\n");
      for( i=1; i<=9; i++ ) {
            printf("%3d",i);
            for( j=1; j<=9; j++ ) {
                  printf("%3d", i*j);
            }
            printf("\n");
      }
}
```

Pointer Data Types

- Use basic data types to define variables:

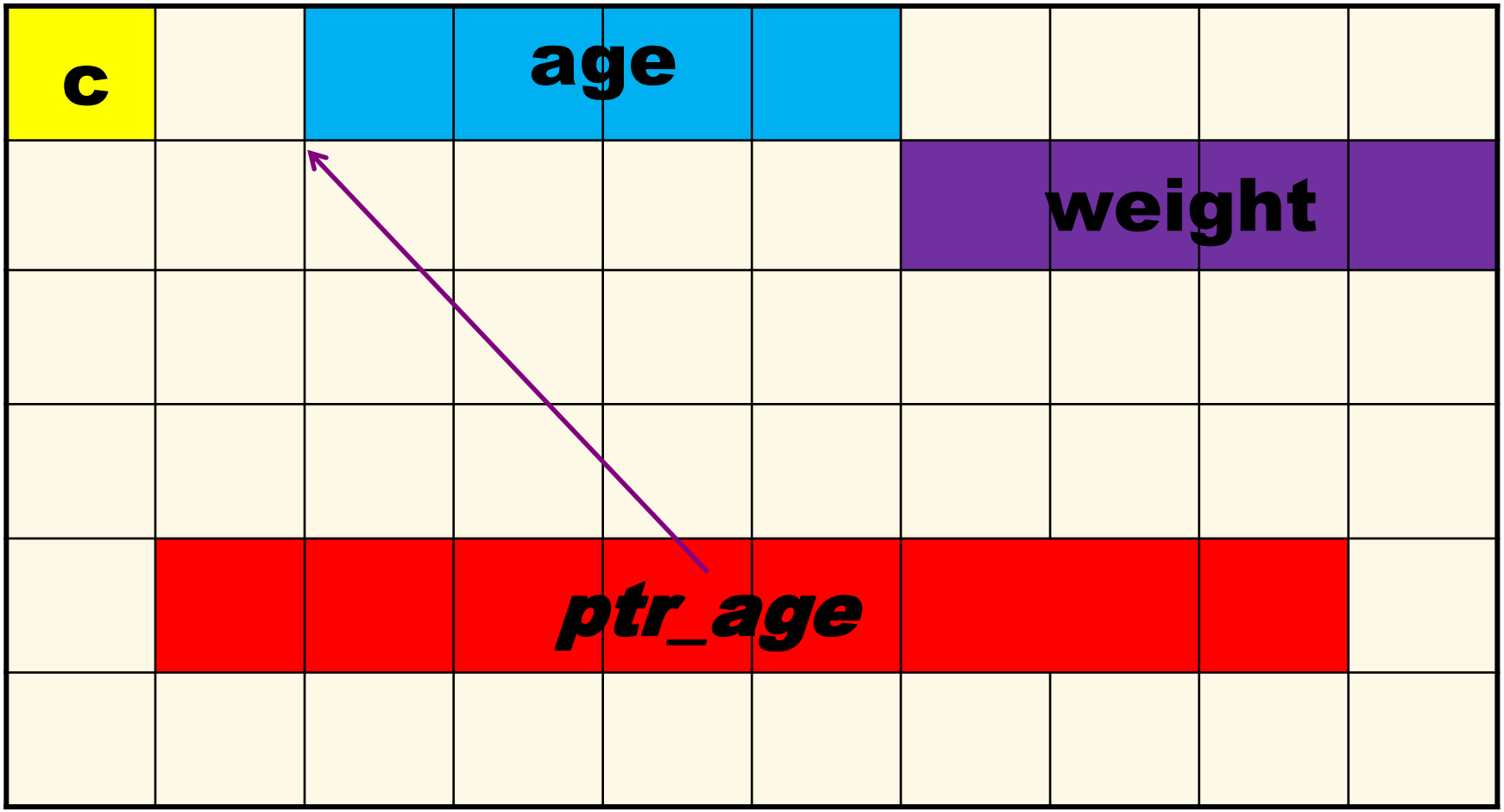
```
char   c = 'A';
int    age = 4;
float  weight = 142.5;
```

- Use pointers to store addresses of variables:

```
char *ptr_c = &c;      // (char*) ptr_c;  wrong
int*   ptr_age = &age;
float* ptr_weight = &weight;
```

Memory View of Variables

To Declare a variable means to reserve memory space for data.



Basic and pointer data types

```
Line 1:  #include <stdio.h>
Line 2:  int main()
Line 3:  {
Line 4:          char c='A', *ptr_c=&c;
Line 5:          int age=4, *ptr_age=&age;
Line 6:          float weight=50.0, *ptr_weight=&weight;
Line 7:
Line 8:          printf("char size:%d\n", sizeof(c));
Line 9:          printf("int size:%d\n", sizeof(age));
Line 10:         printf("float size:%d\n", sizeof(weight));
Line 11:         printf("double size:%d\n", sizeof(double));
Line 12:         printf("address size:%d,%d\n", sizeof(ptr_c), sizeof( int* ) );
Line 13:         printf("c address:%u\n", ptr_c);
Line 14:         printf("c value:%c\n", *ptr_c);
Line 15:         printf("age address:%u\n", ptr_age);
Line 16:         printf("age value=%d,%d\n", age, *ptr_age);
Line 17: }
```

Function Terminology

- Function *prototype* describes how a function is called

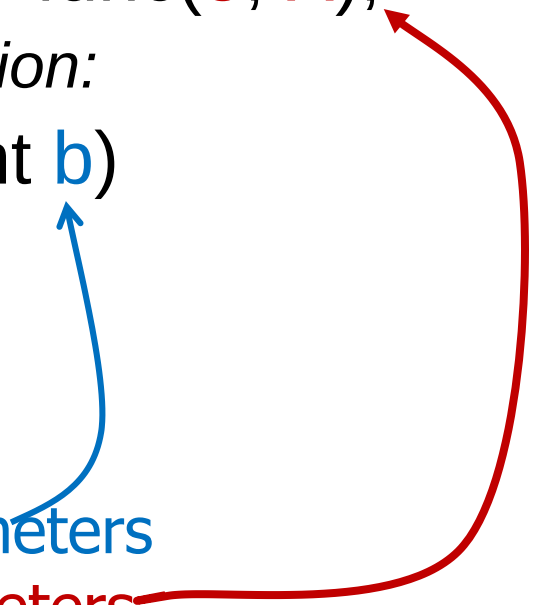
```
int func(int a, int b);  // a,b not necessary
```

- Function *call*:

```
result = func(5, X);
```

- Function *implementation:*

```
int func(int a, int b)
{
   …
}
```



  - Formal parameters
  - Actual parameters
  - Actual parameters must match with formal ones in ***order***, ***number*** and ***data type***.

Parameter Passing

- Call by *value*
  - formal parameter receives the *value* of the actual parameter
  - function can **NOT** change the value of the actual parameter

    example: printf("…", x, y)

- Call by *reference*
  - actual parameters are ***pointers***
  - formal parameter receives the ***value*** of the actual parameter
  - function can change the value of the actual parameter

    example: scanf("…", &x, &y)

Parameter Passing Example

```
Line 1:  #include <stdio.h>
Line 2:  void swap(int, int);
Line 3:  int main()  {
Line 4:      int a=10, b=20;
Line 5:      swap(a, b);
Line 6:      printf("a=%d, b=%d\n",a,b);
Line 7:  }
Line 8:  void swap(int x,  int y) {
Line 9:      int t;
Line 10:     t=x;    x=y;    y=t;
Line 11: }
```

Output:

a=10, b=20

```
#include <stdio.h>
void swap(int *,  int *);
int main()  {
    int a=10, b=20, *pa=&a, *pb=&b;
    swap(pa, pb);
    printf("a=%d, b=%d\n",a,b);
}
void swap(int *x,  int *y) {
    int t;
    t=*x;    *x=*y;    *y=t;
}
```

Output:

a=20, b=10

Thank you!

Exercise

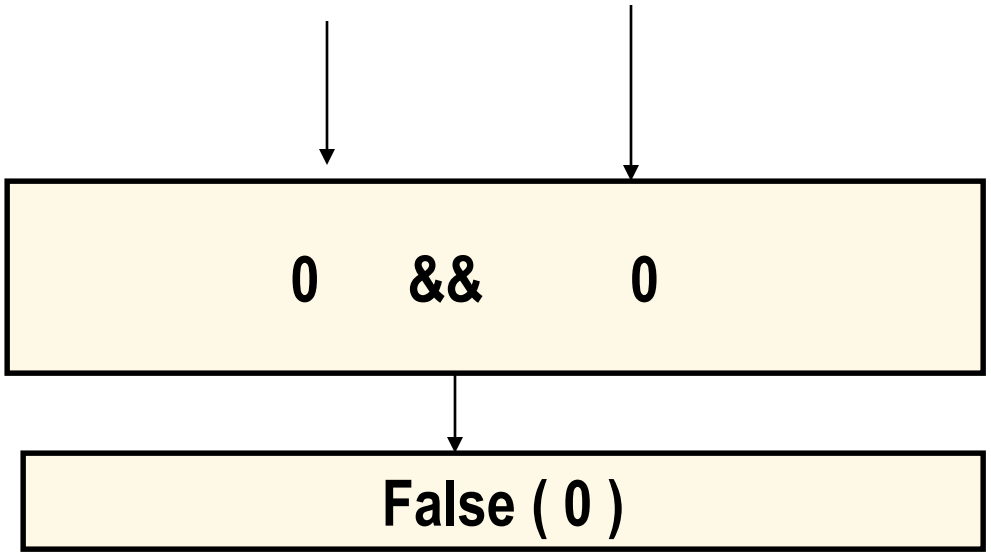
- **Suppose that i, j, and k are integer variables whose values are 1, 2 and 3, respectively.**

Expression	Value	Interpretation
i < j	1	true
(i + j) > = k	1	true
(j + k) > (i + 5)	0	false
k!=3	0	false
j = = 2	1	true

Logic AND

- The output of AND operation is **TRUE** if BOTH the operands are true.

Example: **(8 < 7) && (6 > 7)**



Logical OR

- The result of a *logical or* operation will be true if either operand is true or if both operands are true.

(8 < 7) | | (6 > 7) is false

(8 > 7) | | (6 > 7) is true

(8 > 7) | | (6 < 7) is true

Logical NOT

- The Logical NOT (!) is a unary operator. It negates the value of the logical expression or operand.

- If value of X = 0 ! X = ?
  ! X = 1

- ! (5 < 6) || (7 > 7) = ???
  ! (1) || (0) = ! 1 = 0 -> false

- ! (5 > 3) = ??
  -> 0 -> false

- !(34 >= 765) = ??
  -> 1 -> True.

Exercise

x = 10 and y = 25

(x > = 10) && (x < 20)	True
(x >= 10) && (y < = 15)	False
(x = = 10) && (y > 20)	True
(x==10) || (y < 20)	True
(x ==10) &&(! (y < 20))	True

Exercise

- **Suppose that**

  j = 7, an integer variable

  f = 5.5, a float variable

  c = 'w'

  Interpret the value of the following expressions:

(j >= 6) && (c = = 'w')	**1**
(j >= 6) | | (c = = 'w')	**1**
(f < 11) && (j > 100)	**0**
(c ! = 'p') | | ((j + f) <= 10)	**1**
f > 5	**1**
!(f > 5)	**0**
j < = 3	**0**
!(j <= 3)	**1**
j > (f +1)	**1**
!(j > (f +1))	**0**

- Suppose that

  j = 7, an integer variable

  f = 5.5, a float variable

  c = ‘w’

  Interpret the value of the following expressions:

j + f <= 10	0
j >= 6 && c = = ‘w’	1
f < 11 && j > 100	0
!0 && 0 | | 0	0
!(0 && 0) | | 0	1