LECTURE-03

OPERATORS

IDENTIFIERS

Identifier refers to the name of variables, functions and arrays.

Rules for constructing Identifiers name in C:

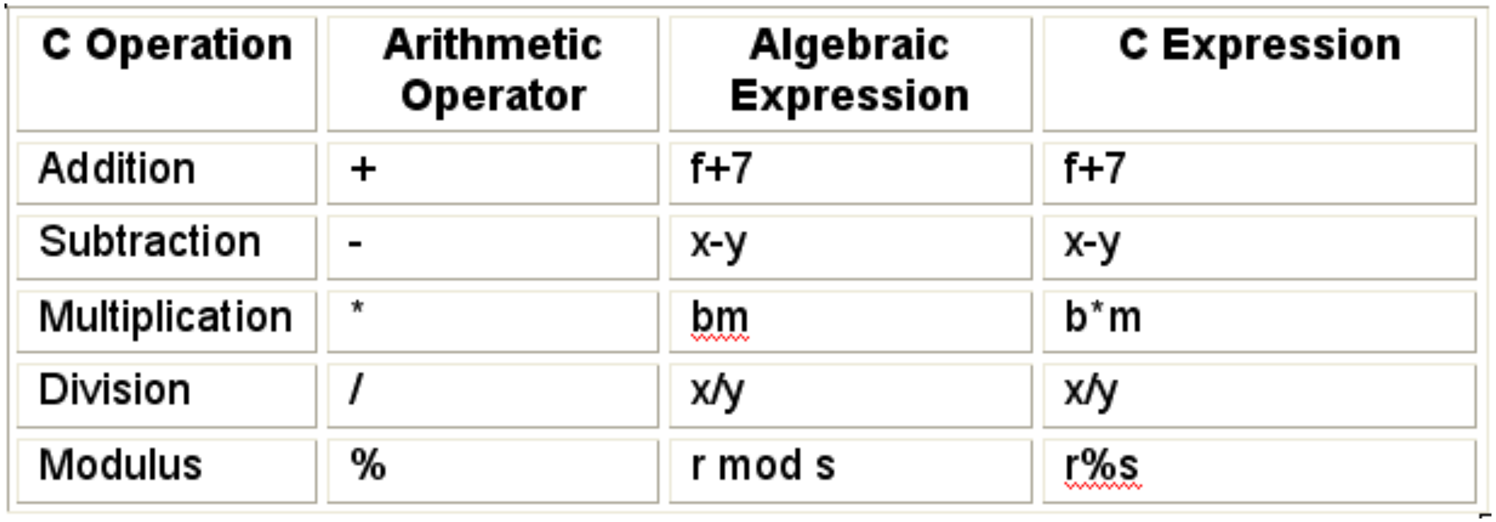
- Identifier is consisting of letters, digits and underscore
- Identifier does not begin with digits.
- Punctuation and special characters aren't allowed except underscore.
- Identifiers should not be keywords.

OPERATOR

- It is a symbol that tells the computer to perform certain mathematical or logical manipulations.
- C operators can be classified into a number of categories:
  - Arithmetic operators
  - Relational operators
  - Logical operators
  - Assignment operators
  - Increment and decrement operators
  - Conditional operators
  - Bitwise operators
  - Special operators

ARITHMETIC OPERATORS

Most C programs perform arithmetic calculations



C Operation	Arithmetic Operator	Algebraic Expression	C Expression
Addition	+	f+7	f+7
Subtraction	-	x-y	x-y
Multiplication	*	bm	b*m
Division	/	x/y	x/y
Modulus	%	r mod s	r%s

ARITHMETIC OPERATORS

- The arithmetic operators are all *binary operators*
- The expression 3+7 contains binary operator + and two *operands* 3 and 7
- Integer division yields an integer result. 17/4 is 4
- The modulus operator means the remainder after a division operation. 17%4 is 1
- The % can not be used with real operand.

A SIMPLE PROGRAM

```
int main(){
    int months,days;
    printf(“enter days\n”);
    scanf(“%d”,&days);
    months= days/30;
    days=days%30;
    printf(“months=%d and days=%d”,months,days);
    return 0;
}
```

RULES OF OPERATOR PRECEDENCE

1. Expressions or portion of expressions contained in pairs of parentheses are evaluated first. In case of nested parentheses, the inner most parentheses will be evaluated first.

RULES OF OPERATOR PRECEDENCE

2. Multiplication, division and modulus operations are evaluated next. If an expression contains several *, / or % operations, evaluations proceed from left to right. These operators are on the same level of precedence.

RULES OF OPERATOR PRECEDENCE

3. Addition and subtraction are evaluated last. If an expression contains several + or - operations, evaluations proceed from left to right.

BRAIN STORM I

- 😐 $m=(a+b+c+d+e)/5$
- 😐 $m=a+b+c+d+e/5$
- What are the differences here?

BRAIN STORM II

😐 y=mx+c

- How do you write this expression in C? Any parenthesis required?

DECISION MAKING: RELATIONAL OPERATORS

Executable C expressions either do *actions* (calculations or input or output) or make a *decision*

For example, in a program where the user will put his mark and want to know if he passed or failed, you will have to do three things-

1. Take the mark from the user. *(Action)*

2. Compare the mark with a predefined pass mark range. *(Decision)*

3. Provide that decision to user. (*Action*)

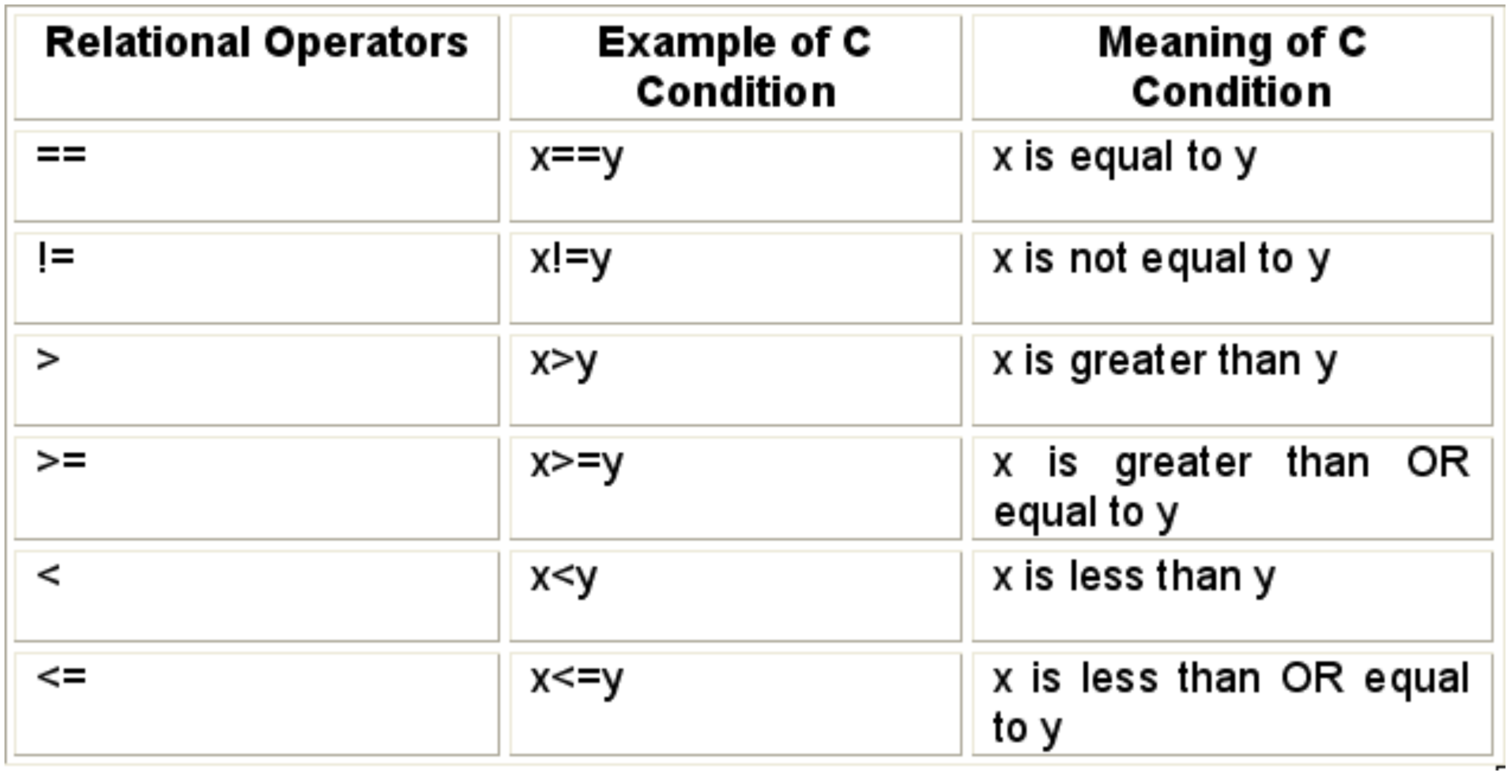
DECISION MAKING: RELATIONAL OPERATORS

- The decision in C is done by a control structure called *if*
- It allows a program to make a decision based on the truth or falsity of some statement of fact called *condition*
- If the condition is true, *the body statement* will be executed otherwise not- it will execute the next statement after the *if* structure

DECISION MAKING: RELATIONAL OPERATORS

- Conditions in *if* structures are formed by using *relational operators*
- The relational operators have the same level of precedence and they associate left to right
- Only *equality operators* have a lower level of precedence than others and they also associate left to right

DECISION MAKING: RELATIONAL OPERATORS



Relational Operators	Example of C Condition	Meaning of C Condition
==	x==y	x is equal to y
!=	x!=y	x is not equal to y
>	x>y	x is greater than y
>=	x>=y	x is greater than OR equal to y
<	x<y	x is less than y
<=	x<=y	x is less than OR equal to y

DECISION MAKING: RELATIONAL OPERATORS

- There is a subtle difference between == and =. In case of =, it is associated from right to left. a=b means the value of b is assigned to a

DECISION MAKING: RELATIONAL OPERATORS

```
#include <stdio.h>
int main(){
    int num1, num2;
    printf("Enter two numbers to compare: ");
    scanf("%d %d", &num1, &num2);
    if(num1 == num2)
      printf("%d and %d are equal", num1, num2);
    if(num1 != num2)
      printf("%d and %d are not equal", num1, num2);
    if(num1 < num2)
      printf("%d is less than %d", num1, num2);
    if(num1 > num2)
      printf("%d is greater than %d", num1, num2);
    if(num1 <= num2)
      printf("%d is less than or equal to %d", num1, num2);
    if(num1 >= num2)
      printf("%d is greater than or equal to %d", num1, num2);
    return 0;
}
```

LOGICAL OPERATORS

Logical AND	&&
Logical OR	||
Logical NOT	!

- The logical operator && and || are used for testing more than one conditions and make decisions.
- a>b && x==10

LOGICAL OPERATOR(TRUTH TABLE)

Op1(a>b)	Op2(x==10)	Op1 && Op2	Op1||Op2
1	1	1	1
1	0	0	1
0	1	0	1
0	0	0	0

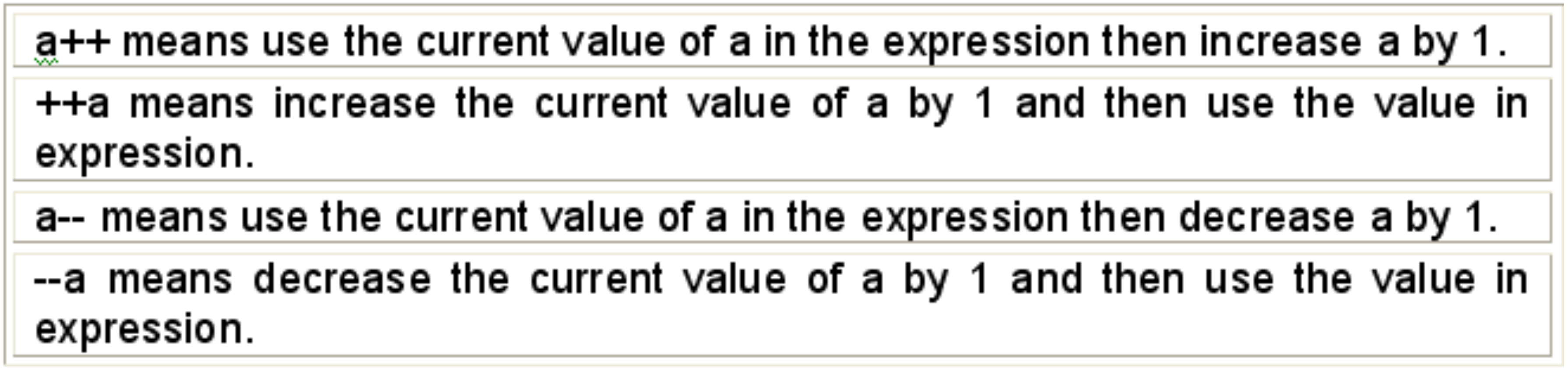
ASSIGNMENT OPERATOR

Statement with simple assignment operator	Statement with shorthand operator
a=a+1	a+=1
a=a-1	a-=1
a=a*(n+1)	a*=(n+1)

INCREMENT AND DECREMENT OPERATORS

- C provides the unary increment operator ++ and unary decrement operator —
- If a variable a is incremented by 1, the increment operator ++ can be used instead of expression a=a+1
- But the meaning differs based on the place of the operator to the variable

INCREMENT AND DECREMENT OPERATORS



a++ means use the current value of a in the expression then increase a by 1.
++a means increase the current value of a by 1 and then use the value in expression.
a-- means use the current value of a in the expression then decrease a by 1.
--a means decrease the current value of a by 1 and then use the value in expression.

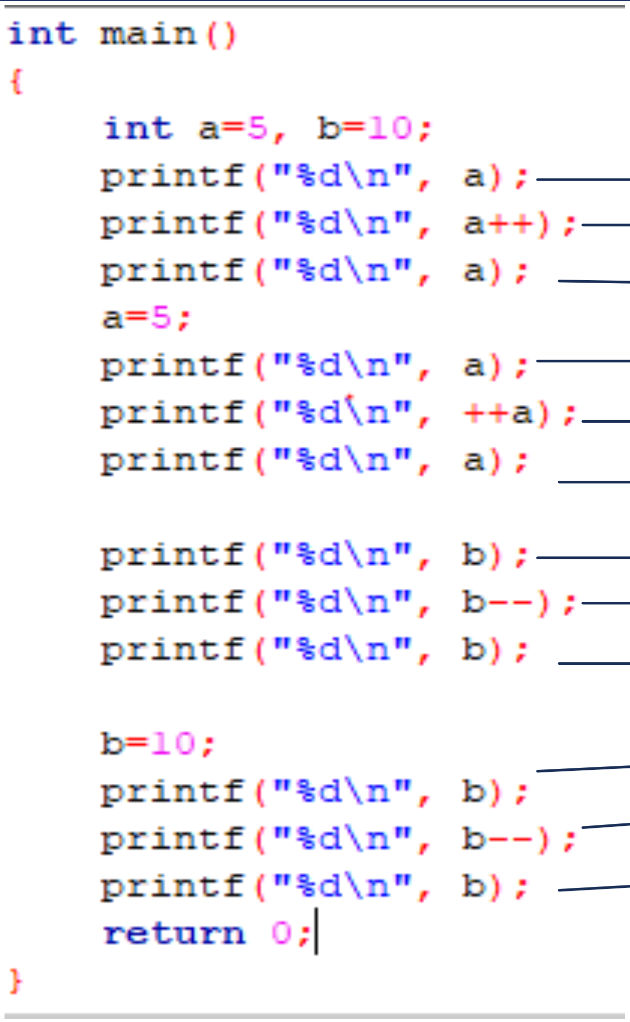
INCREMENT OPERATOR AND DECREMENT OPERATOR

- Increment operator:
  - m=5;
  - y=++m;
  - A prefix operator first add 1 to the operand and then the result is assigned to the variable on the left.
  - y=m++;
  - A postfix operator first assign the value to the variable on the left and then increments the operand.

INCREMENT AND DECREMENT OPERATORS

- If unary increment/ decrement operators are placed in front of the variable, they are called preincrement/ predecrement operators
- otherwise they are called postincrement/ postdecrement operators.

INCREMENT AND DECREMENT OPERATORS

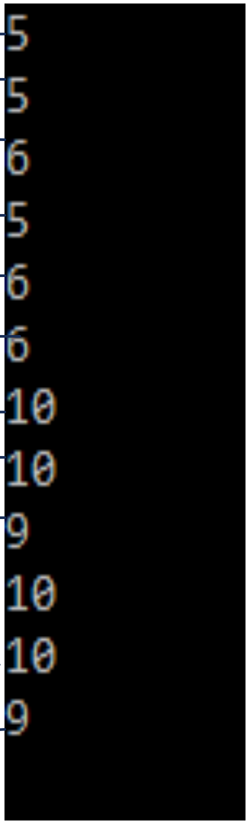


```
int main()
{
    int a=5, b=10;
    printf("%d\n", a);
    printf("%d\n", a++);
    printf("%d\n", a);
    a=5;
    printf("%d\n", a);
    printf("%d\n", ++a);
    printf("%d\n", a);

    printf("%d\n", b);
    printf("%d\n", b--);
    printf("%d\n", b);

    b=10;
    printf("%d\n", b);
    printf("%d\n", b--);
    printf("%d\n", b);
    return 0;
}
```

Output



```
5
5
6
5
6
6
10
10
9
10
10
9
```

INCREMENT AND DECREMENT OPERATORS

```
a=15;
b=10;
c=++a – b;
d=b++ +a;
```

What is the value of c and after executing above statements line by line.

CONDITIONAL OPERATORS

- This operator replaces certain statements of the if-then-else.
- The operator ? takes the general form
- ✓ Condition ? Expression 1 : expression 2;

Example:

```
a= 10, b=9;
c=(a >b)? a : b ;
```

BITWISE OPERATOR

Operator	meaning
&	Bitwise AND
|	Bitwise OR
^	Bitwise XOR
<<	Shift left
>>	Shift right
~	1's complement

- It is used for manipulating data at bit level
- Used for testing bits or shifting them right or left.

SPECIAL OPERATOR

- Comma operator
- Sizeof operator:
  - It returns the no of bytes the operand occupies.
  - m=sizeof(sum);
  - n=sizeof(int);

TYPE CONVERSIONS IN EXPRESSIONS

- Automatic type conversion
- Casting a value

AUTOMATIC TYPE CONVERSION

- If the operands are of different type the lower type is automatically converted to the higher type before the operation proceeds. The result is of higher type.
- Ex:
  - int i,x;
  - float f;
  - double d;
  - long int l;
- X= l / i + i*f –d;

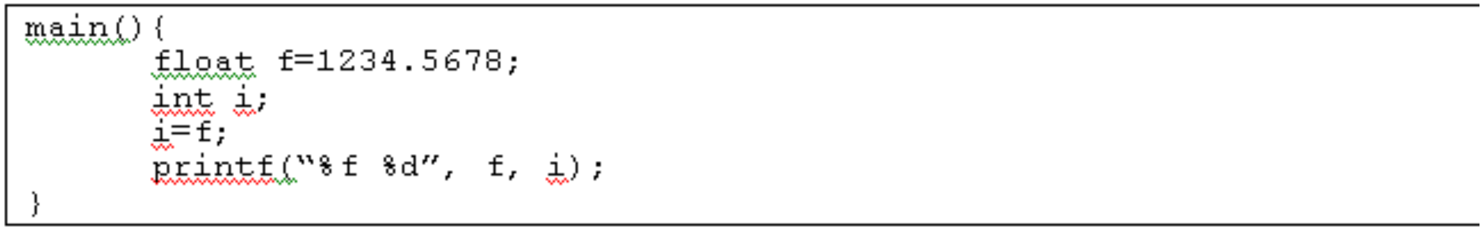
ORDER OF TYPE

- long double
- double
- float
- unsigned long
- long
- Unsigned
- int
- All short and char are automatically converted to int
- The final result of an expression is converted to the type of the variable on the left of the assignment sign before assigning the value to it.

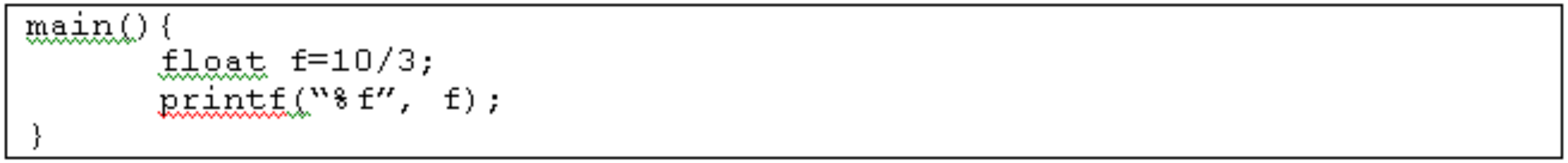
SOME PROBLEMS DURING CONVERSION

- float to int causes truncation of the fractional part.
- double to float causes rounding of the digits.
- long int to int causes dropping of the excess higher order bits.

TYPE CONVERSION IN ASSIGNMENT



```
main(){
        float f=1234.5678;
        int i;
        i=f;
        printf(“%f %d”, f, i);
}
```



```
main(){
        float f=10/3;
        printf(“%f”, f);
}
```

TYPE CAST

- Sometimes it is necessary to force a type conversion that is different from automatic conversion.
- you want to use a variable in modulus operation, so you have declared it as an integer since modulus operation never accepts variables other than integers
- But you may require using the variable in division as well which may produce a floating point result
- You can use type cast this time

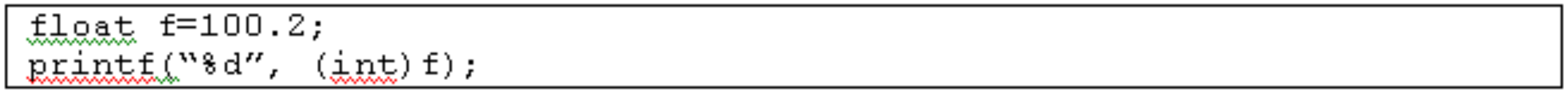
PROGRAM

```
void main(){
    int a=8;
    int b=3;
    int mod;
    float result;
    mod=a%b;
     result=a/b;
    printf(“mod=%d”,mod);
    Printf(“result=%f”,result);
}
```

PROGRAM

```
void main(){
    int a=8;
    int b=3;
    int mod;
    float result;
    mod=a%b;          ←
     result=(float)a/b;          ←
    printf("mod=%d",mod);
    Printf("result=%f",result);
}
```

TYPE CAST



```
float f=100.2;
printf("%d", (int)f);
```

❖ you cannot cast a variable that is on the left side of an assignment statement.

```
int num;
(float)num=123.45;
```

HOMEWORK

- Operator Precendence
- Storage Types – auto, register, extern, static
- I/O