Lecture – 6

Operator precedence and associativity:

1. ( ) , [] → Left to right
2. + + (postfix) , - - (postfix) → Right to left
3. ! (not) , ~ (1's complement) , + (unary) , - (unary) , + + (prefix) , - - (prefix) , & (address) , * (indirection) , sizeof → Right to left
4. * , / , % (modulus)→ Left to right
5. + (binary) , - (binary) → Left to right
6. << (shift left) , >> (shift right) → Left to right
7. < , < = , > , > = → Left to right
8. = = , ! = → Left to right
9. & (bitwise AND) → Left to right
10. ^ (bitwise XOR) → Left to right
11. | (bitwise OR) → Left to right
12. && (logical AND) → Left to right
13. || (logical OR) → Left to right
14. ? : (a ? x : y) → Right to left
15. = , * = , / = , % = , + = , - = , & = , ^ = , | = , << = , >> = → Right to left
16. , (comma) → Left to right

Dealing with expressions:

Example 1:

```
void main(void)
{
int x=2,n=2;
x=n++;
printf("%d",x);
x=++n;
printf("%d",x);
}
```

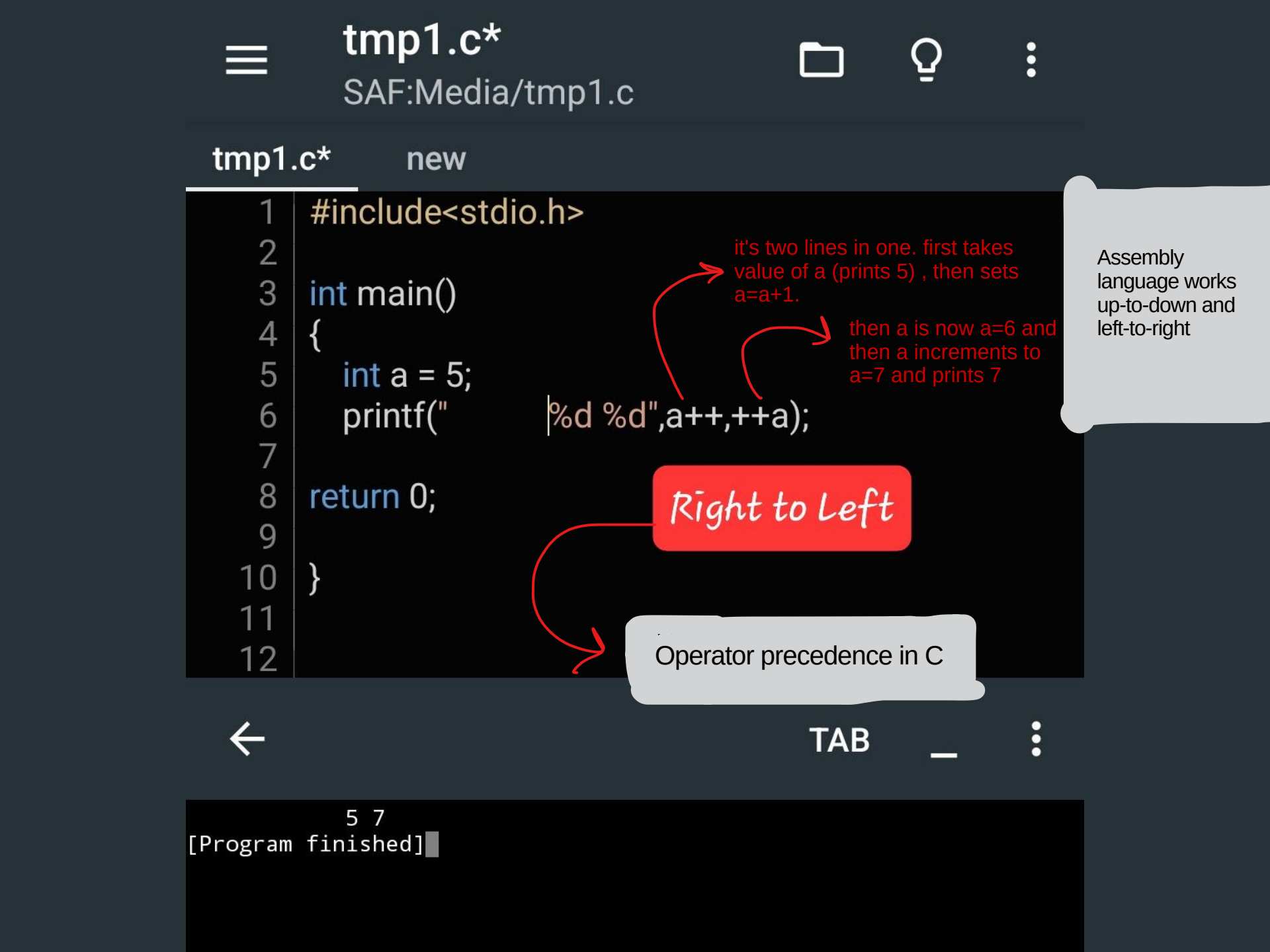
24

Example 2:

```
void main(void)
{
int x=2, y=3;
x*=y;
printf("%d",x);
x=x*y;
printf("%d",x);
x*=y+1;
printf("%d",x);
}
```

61872

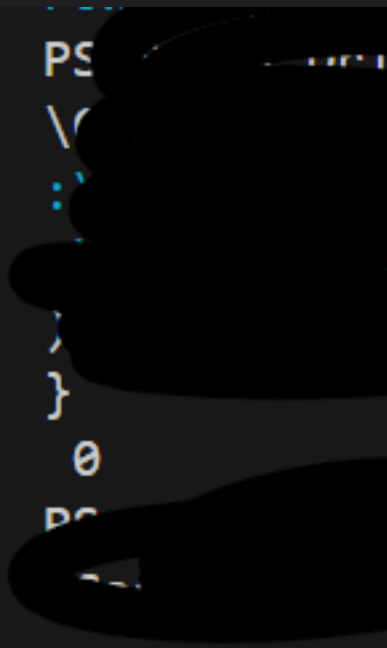
void main(void)


tmp1.c*
SAF:Media/tmp1.c
tmp1.c*
new
#include<stdio.h>
int main()
{
int a = 5;
printf(" %d %d",a++,++a);
return 0;
}
it's two lines in one. first takes value of a (prints 5) , then sets a=a+1.
then a is now a=6 and then a increments to a=7 and prints 7
Assembly language works up-to-down and left-to-right
Right to Left
Operator precedence in C
TAB
5 7
[Program finished]

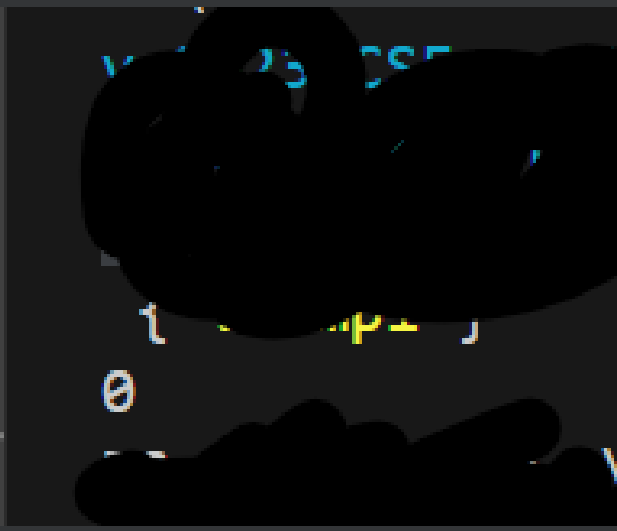
```c
int main()
{
    int x = 1;
    // if(x--){printf("%d",x);}

    if (x--)
    {
        printf("99");
        //the value for if() is true
        //and in the sequence of x
    }
    printf(" %d", x); // the current value

```

```
99 0
```

```c
int main()
{
    int x = 1;
    // if(x--){printf("%d",x);}

    if (--x)
    {
        printf("99");
        //the value for if() is true
        //and in the sequence of x

        //output 0 becuase if() is skipped
    }
    printf(" %d", x); // the current value

    return 0;
}
```



```c
int main()
{
    int x = 1;
    if(x--){printf("%d",x);}
}
```



How does the increment operator work in an if statement?

Asked 10 years, 1 month ago Modified 3 years, 10 months ago

Viewed 40k times

4

```c
#include <stdio.h>

int main()
{
    int x = 0;

    if (x++)
        printf("true\n");
    else if (x == 1)
        printf("false\n");
    return 0;
}
```

Output:

```
false
```

Why is the output false?

`x++` is post increment; this means that the value of `x` is used then it is incremented. If it is so, then `x=0` should be used and the answer should be true.

c if-statement post-increment

Follow Academic calender

printf("%d", x); // 0

Q. int x = 1;

if(x--) → 0

1 ← printf("1"); // 1

if(x--) → -1

0 ← printf("2");

else if(x--) → -2

1 ← printf("3"); // 3

if(x--) → -3

1 ← printf("4"); // 4

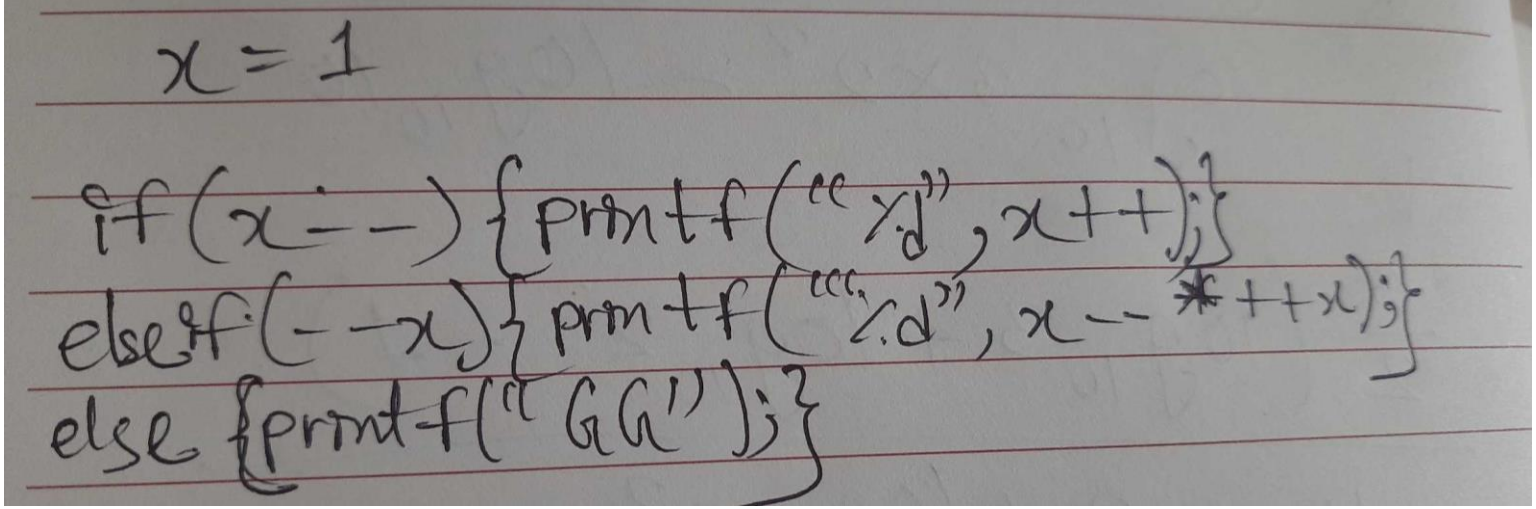
printf("%d", x); // -3

statement: x--; (or) x = x - 1;

expression

without curly bracket only the first printf will be counted. ";" declares a stop

⇒ 1 3 4 -3




```c
#include <stdio.h>
int main()
{
    printf("\n\n\n");

    int x = 1;
    if (x--)
    {
        printf("%d", x++);
    }

    printf("\n\n\n");

    // if the other one was true
    x = 1;
    if (x--) // (--x) would be false
    {
        printf("%d", x--*++x);
    }

    printf("\n\n\n");
    return 0;
}
```

```
PROBLEMS    OUTPUT    DEBUG CONSOLE

0


0
```