CS101C: Introduction to Programming (Using C)

Autumn 2025

Nikhil Hegde
Achyut Mani Tripathi

Week3: Operator precedence, accepting input from terminal, if-else

So far..

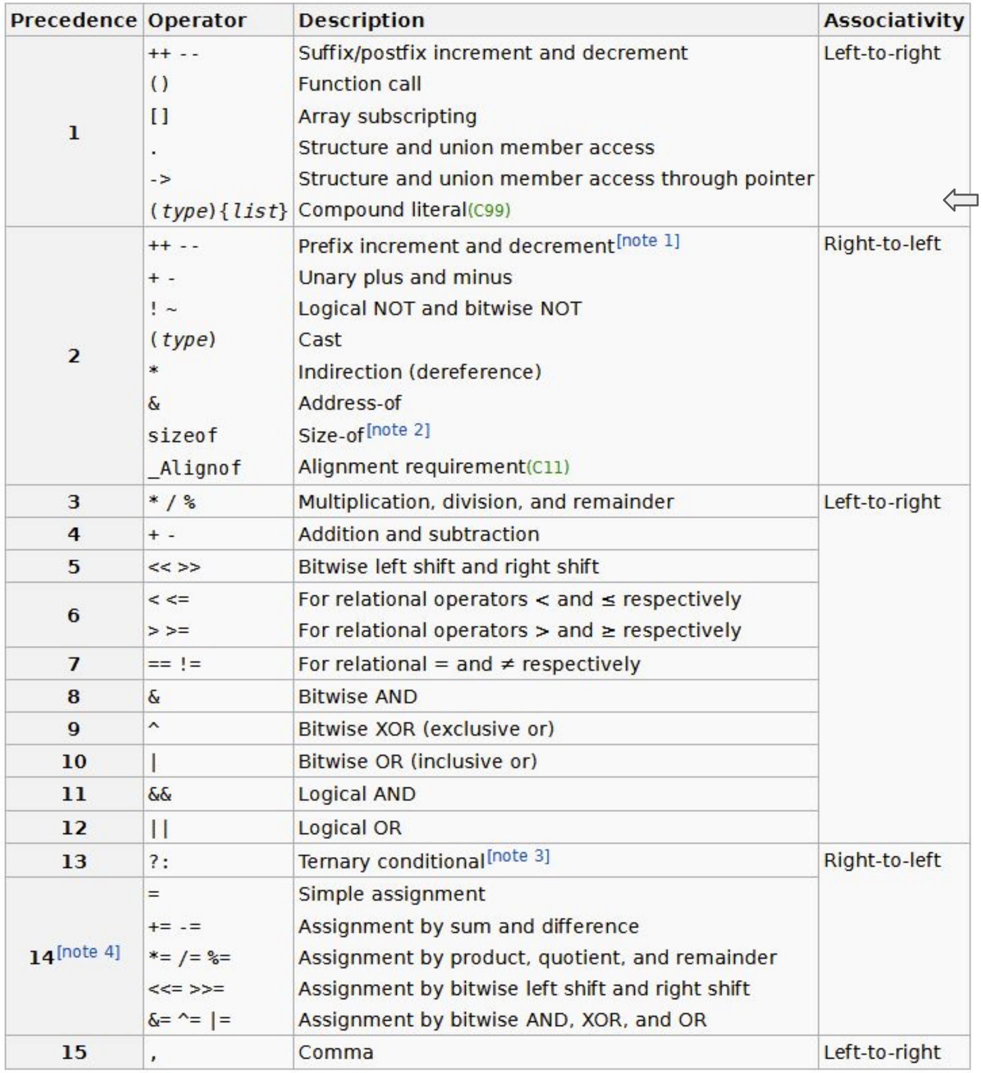
- Printing on the terminal (printf)
- Data Types (int, float, double, char),
- Constants, Variables and their initialization using constants
- Operators and related background (bit representation)
    - Arithmetic (+, -, *, /, %)
    - Relational (==, !=, >, <, >=, <=)
    - Assignment (=, +=, -=, *=, /=, %, <<=, >>=, &=, ^=, |=)
    - Increment / Decrement (++, --)
    - Special: ternary, sizeof, comma
    - Logical (&&, ||, !)
    - Bitwise (&, |, ^, ~, <<, >>)

Today’s class (18/8/2025)

- Operator precedence
- Accepting input from terminal
- C program to demonstrate:
  - Operator precedence
  - Accept input from terminal using scanf

Operator Precedence and Associativity

source: https://en.cppreference.com/w/c/language/operator_precedence.html



Precedence	Operator	Description	Associativity
1	++ --	Suffix/postfix increment and decrement	Left-to-right
	()	Function call	
	[]	Array subscripting	
	.	Structure and union member access	
	->	Structure and union member access through pointer	
	(*type*){*list*}	Compound literal(C99)	
2	++ --	Prefix increment and decrement[note 1]	Right-to-left
	+ -	Unary plus and minus	
	! ~	Logical NOT and bitwise NOT	
	(*type*)	Cast	
	*	Indirection (dereference)	
	&	Address-of	
	sizeof	Size-of[note 2]	
	_Alignof	Alignment requirement(C11)	
3	* / %	Multiplication, division, and remainder	Left-to-right
4	+ -	Addition and subtraction	
5	<< >>	Bitwise left shift and right shift	
6	< <=	For relational operators < and ≤ respectively	
	> >=	For relational operators > and ≥ respectively	
7	== !=	For relational = and ≠ respectively	
8	&	Bitwise AND	
9	^	Bitwise XOR (exclusive or)	
10	|	Bitwise OR (inclusive or)	
11	&&	Logical AND	
12	||	Logical OR	
13	?:	Ternary conditional[note 3]	Right-to-left
14[note 4]	=	Simple assignment	
	+= -=	Assignment by sum and difference	
	*= /= %=	Assignment by product, quotient, and remainder	
	<<= >>=	Assignment by bitwise left shift and right shift	
	&= ^= |=	Assignment by bitwise AND, XOR, and OR	
15	,	Comma	Left-to-right

```
a=10, b=20, c=5. Evaluate:

d=a+b*c

d=a*b/c

d=b<a*c

d+=a=b

d+=a?b:c;

d=a+-b
```

challenge Q: `d=++a+-b;?`

Sample Program - Operator Precedence

```
int main(){
    int a=10, b=20, c=5, d;
    d=a+b*c;
    printf("result of d=a+b*c: %d\n", d);
    d=a*b/c;
    printf("result of d=a*b/c: %d\n", d);
    d=b<a*c;
    printf("result of d=b<a*c: %d\n", d);
    d=0;
    d+=a?b:c;
    printf("result of d+=a?b:c %d\n", d);
    d=0;
    d+=a=b;
    printf("result of d+=a=b: %d. current value of a: %d\n", d, a);
    a=10;
    d=a+-b;
    printf("result of d=a+-b: %d.\n", d);
}
```

Scanf - a way to accept user input

```
int main() {
    int a;
    scanf(“%d”,&a);
    printf(“%d”,a);
}
```

- Reads user input from terminal (`stdin`) and stores in variables
- **&** - address of operator.
- **Arguments of scanf**: 1st = string, 2nd and subsequent (if present) = address of variables. How do you know if second and subsequent arguments are present?

Today’s class (20/8/2025)

- Control-flow: `if-else`
- C program to demonstrate:
  - How the flow of execution can be controlled using the `if-else` construct

Control flow with `if-else`

- So far you have seen flow of control, where an instruction on a line executed after the instruction on the previous line.
- *The* `if-else` *construct in C changes that* i.e. the previously executed instruction need not necessarily be the one on the previous line.
- `E.g.` Execution of lines

- 4 not preceded by 3
- 6 not necessarily after 4

```
0: scanf(“%d”,&age);

1: if(age<18){

2: printf(“cannot drive”);

3: } else {

4: printf(“drive safe”);

5:}

6: printf(“bye”);
```

Control flow with `if-else`

- Curly braces {} can be omitted if you have a single statement within if-block / else-block
- The `else` part can be skipped. But if you have `else` part, you must have `if` part
- **Meaning of `if-else`:**

  If the condition is true, execute the if-block

  Otherwise execute the else-block

```
if(condition){
    statement1;
    ..
    statementn;
} else {
    statement1;
    ..
    statementn;
}
```

Which operator allowed conditional execution of an expression?

Control flow with `if-else`

- Can have `if-else` within another `if` and/or `else`
- New lines can be skipped. But having new lines makes your code readable.

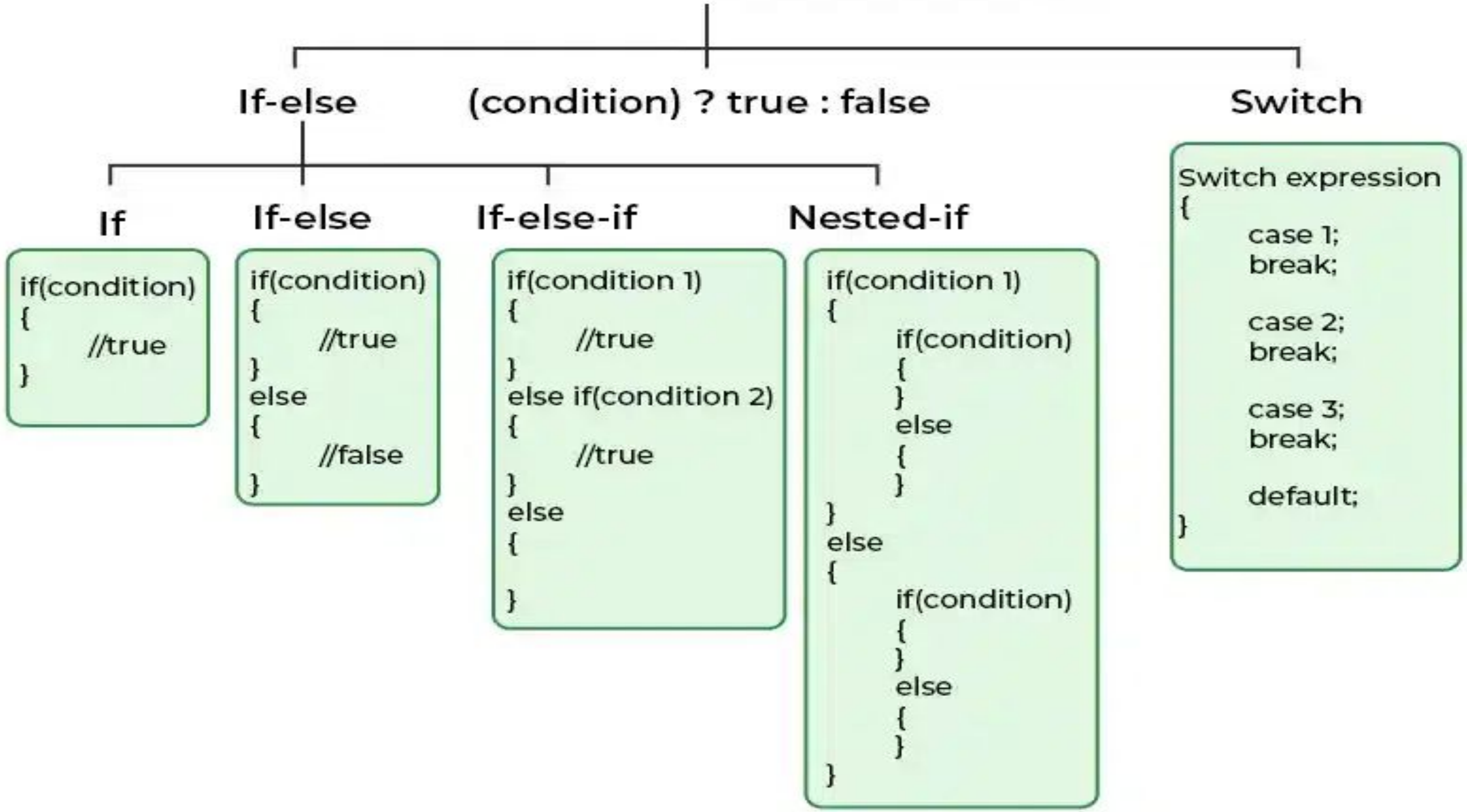
```
if(a<10){
   if(a>5){x=10;}
   else{x=100;}
} else {
   if(a>5){x=20;}
   else{x=200;}
}
```

Control flow with `if-else`

- There is `else-if` as well:
- The last else part is optional

```
if(a < 10){

   x=10;

} else if (a < 20){

   x=100;

} else if (a<30){

   x=1000;

} else {

   x=10000;

}
```

Conditional Statements in C



```
if(condition)
{
        //true
}
```

```
if(condition)
{
        //true
}
else
{
        //false
}
```

```
if(condition 1)
{
        //true
}
else if(condition 2)
{
        //true
}
else
{

}
```

```
if(condition 1)
{
        if(condition)
        {
        }
        else
        {
        }
}
else
{
        if(condition)
        {
        }
        else
        {
        }
}
```

```
Switch expression
{
        case 1;
        break;

        case 2;
        break;

        case 3;
        break;

        default;
}
```

Recap: if-else

```
if(a<10)
    if(a<5)
        printf(“a is less than 5\n”);
    else
        printf(“a is not less than 5\n”);
```

- Which `if` does `else` get attached to?

Recap: if-else

```
if (n > 0)
    if (n > 10) {
        printf("n is greater than 10");
    }
else
    printf("n is negative\n");
```

- Which `if` does `else` get attached to?

Recap: else-if

```
if(a<15)
    if(a<5)
        printf(“a is less than 5\n”);
    else if(a<8)
        printf(“a is >=5 but < 8\n”);
    else if(a<12)
        printf(“a is >=8 but < 12\n”);
else
    printf(“a is >=15)
```

Syntax:

```
if(expression1){
} else if(expression2){
} else if(expression3){
} else {
}
```

What message gets printed when a is 15?

Today’s class (22/8/2025)

- Control-flow: `switch-case`
- C program to demonstrate:
  - The flow of execution using the switch`-case` construct

Control flow with `switch-case`

- `if-else` gives you two-way branching
- with `else-if` you get multi-way branching
- `switch` statement in C also gives you multi-way branching.

```
switch(class){

    case 1: printf(“MON”);

    case 2: printf(“WED”);

    case 3: printf(“FRI”);

    default: printf(“no
class”);

}
```

`switch` statement - syntax

```
switch(expression){

    case constant_expression1: statements;

    case constant_expression2: statements;

    default: statements;

}
```

- **`case`** is followed by integer valued constants OR constant expressions
- **`default`** is optional
- for single set of statements, can attach multiple **`case`** constant_expression:
- Meaning: if expression matches one of the constant expressions, execution begins at the corresponding statements.

Demo - `switch` statement

- Write a C program:

  - accept a digit from terminal.

  - using switch statement, print what digit was entered

    E.g. if you entered 9, your output should look like this:

      “You entered the digit NINE”

Sample Program - switch-case

```
int main(){

        int digit;

        printf("Enter a digit\n");

        scanf("%d",&digit);

        switch(digit){

                case 0:

                case 1:

                case 2: printf("you entered the digit < TWO\n");

                case 3: printf("you entered the digit THREE\n");

                case 4: printf("you entered the digit FOUR\n");

                case 5: printf("you entered the digit FIVE\n");

                case 6: printf("you entered the digit SIX\n");
                case 7: printf("you entered the digit SEVEN\n");
                case 8: printf("you entered the digit EIGHT\n");
                case 9: printf("you entered the digit NINE\n");
                        break;
                default: printf("you entered a non-digit\n");
        }

        printf("end of program\n");
}
```