CPE101 Programming Languages I

Week 2
Variable Concept and Basic Operators

Assist. Prof. Dr. Caner ÖZCAN

Object

- Any accessible thing which takes a memory space is called an **object**.
- An expression should indicate a memory space to be called as an object.
  - a = b+c;
  - d = 100;
- In the above expressions, **a**, **b**, **c** and **d** are all an object.

Object

- **Properties of Objects:** name, value, type, scope, lifetime.
- **Name:** Characters that represent an object.
- **Value:** Information stored in an object. It can be changed at any time.
- **Type:** A property that specifies how a compiler behaves to an object on a process.
  - Most of the programming languages includes object types such as **char**, **integer** and **float**.

Assignment Operator

- Assigns a value to an object. It is showed by an equal sign " = " in C.
- Usage of assignment operator:

```
object = expression;
```

- Examples:

```
a = 23;
b = a * 10;
total = total + b;
```

Left Values (lvalue)

- All expressions that specify object are left values.
- An expression is called as left value if it shows a location in the memory.
- For example, in previous example expression, a and b are the left values.
- But, **a+b** is not a left value. It only represents a number which indicates the sum of a and b.
- For example we can not write, **a+b = c**

Right Value (rvalue)

- Expressions that do not specify objects. They take place on the right side of assignment operator.
- Constants are always right value.
- For example, in an expression **a = 100**; **a** indicates a left value and **100** indicates right value.
- An expression like **100 = a;** is wrong.
- Following expressions have mistakes.

```
20 = ...;       /* mistake */
c - 4 = ...;    /* mistake */
(y) = ...;      /* mistake */
m * 2 = ...;    /* mistake */
```

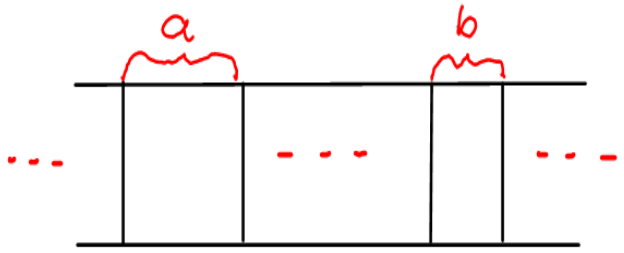
Object Type

- All information that points a memory space or not, is called data.
- Both constants and objects are all data.
- The way that compiler interprets an information stored inside an object depends on the type of that object.
- At the same time, an object type gives information about the amount of memory space that is consumed by the object.

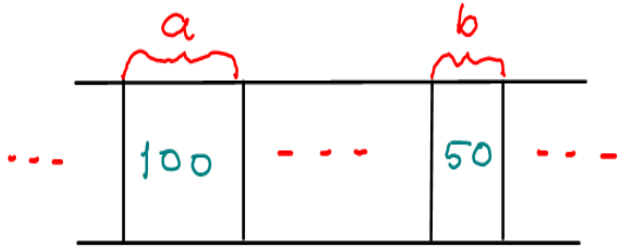
Object Type

- Objects are stored at a location inside the memory.
- For example, objects **"a"** and **"b"** are put in a free location in the memory.
- Memory space they consume depends on their types and can be different.
- **"a"** and **"b"** are only labels that indicate the starting point of a location in the memory.
- An assignment like **a = 100** changes the value in the memory location indicated by related object.
- For example, we have two objects assigned with values **a= 100** and **b = 50**
- An expression like **a = b + 80** only changes the value of a but b is preserved.

Object Type



a ve b nesneleri tanımlandı



sol taraf değeri

sağ taraf değeri (sabit)

a = 100 } değer
b = 50 } atması

a = b + 80

Expression

- An expression is a mathematical formula used for calculation and end with a semicolon ";"
  - (a+b)/4;
  - a*b+c;
- Expressions are formed by Operators
- C operators can be classified as shown below:
  - Assignment Operator (=)
  - Arithmetic Operators (+, -, *, /, %)
  - Arithmetic Assignment Operators (+=, -=, *=, …)
  - Increment and Decrement Operators (++, --)
  - Relational Operators (<, <=, ==, >=, >)
  - Logical Operators (&&, ||, !)

Arithmetic Operators

- The arithmetic operators are all binary operators.
  - For example the expression 3+7 contains the binary operator + and the operands 3 and 7.
- The asterisk (*) indicates multiplication and the percent sign (%) denotes the remainder operator.
- Integer division yields an integer result.
  - For example the expression 7/4 yields 1.

Arithmetic Operators

- C provides remainder operator %, which yields the remainder after integer division.
- The remainder operator is an integer operator that can only be used with integer operands.
- The expression x % y yields the remainder after x is divided by y. Thus 7%4 yields 3.

Arithmetic Operators

Operation	Arithmetic Operator
Addition	+
Subtraction	-
Multiplication	*
Division	/
Remainder	%

Precedence Rules on Arithmetic Operators

ORDER	OPERATOR	OPERATION
1	( )	Paranthesis
2	* / %	Mutiplication Division Remainder
3	+ -	Addition Subtraction

Precedence Rules on Arithmetic Operators

- Expressions within pairs of parentheses are evaluated first.
- Parentheses are said to be highest level of precedence.
- In cases of nested or embedded parentheses such as
  - ((a+b)+c) (**the operators in the innermost pair of parentheses are applied first)**
- Paranthesis in the same level are evaluated from left to right.
- Multiplication, division and remainder comes after parenthesis.
- Addition and subtraction has the same level of precedence, which is lower than the precedence of multiplication, division and remainder operations.

Precedence Rules on Arithmetic Operators

- Multiplication, division and remainder are said to be on the same level of precedence.
- If an expression contains several multiplication, division and remainder operations, evaluation proceeds from left to right.
- If an expression contains several addition and subtraction operations, evaluation proceeds from left to right.
- Remembering rules of precedence can be complex.
- You would better try to use parenthesis in order to specify precedence of operators in expressions.
  - For example: result = (a*b) + (a/b);

Precedence Rules on Arithmetic Operators

- If we want to divide the entire quantity (a+b+c+d+e) by 5.

  m= (a + b + c + d + e) / 5;

- Here, parentheses are required to group the additions because division has higher precedence than addition.
- If the parentheses are omitted we obtain a+b+c+d+e/5. And it would first calculate e/5 then additions.

  z = p * r % q + w / x – y;

- y = a * x * x + b * x + c;

  a = 2, b = 3, c = 7 and x = 5

  y = 2 * 5 * 5 + 3 * 5 + 7

  y = 10 * 5 + 3 * 5 + 7

  y = 50 + 3 * 5 + 7

  y = 50 + 15 + 7

  y = 65 + 7

  y = 72

Arithmetic Assignment Operators

- Arithmetic assignment operators are:

  += -= *= /= %= …

Assignment operator	Sample expression	Explanation	Assigns
Assume: `int c = 3, d = 5, e = 4, f = 6, g = 12;`			
+=	`c += 7`	`c = c + 7`	10 to c
-=	`d -= 4`	`d = d - 4`	1 to d
*=	`e *= 5`	`e = e * 5`	20 to e
/=	`f /= 3`	`f = f / 3`	2 to f
%=	`g %= 9`	`g = g % 9`	3 to g

Unary Increment and Decrement Operators

- result = ++a; → first increment the value of a, then assign it to result (preincrement)
- Same with :

```
a = a+1;
result = a;
```

- result = --a; → first decrement the value of a, then assign it to the result (predecrement)
- Same with:

```
a = a-1;
result = a;
```

Unary Increment and Decrement Operators

- result = a++; → First assign the value of a to result, then increment the value of a (postincerement)
- Same with:

```
result = a;
a = a+1;
```

- result = a--; → First assign the value of a to result, then decrement the value of a (postdecrement)
- Same with:

```
result = a;
a = a-1;
```

- It's important to note here that when incrementing or decrementing a variable in a statement by itself, the preincrement and postincrement forms have the same effect. Same with:

Relational Operators

- Expressions that compare two values and produce either True (1) or False (0) are formed by relational operators.

Relational Operator		
==	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 less than Y
>=	X >= Y	X is greater than or equal to Y
<=	X <= Y	X is less than or equal to Y

Relational Operators

- C does not have an explicit boolean type
  - So integers are used instead. The general rules is:
  - "Zero is false, any non-zero value is true"
- Assume that, a = 1, b = 2, and c = 3

Expression	Result	Value
a < b	True	1
(a + b) >= c	True	1
(b + c) > (a + 5)	False	0
c != 3	False	0
b == 2	True	1

Relational Operators

- Used to combine relational expressions that are either True (1) or False (0)
- Their result is again "True" or "False«
- If a number is interpreted in logical way, the rule is:
  - 0 → False
  - No zero positive or negative numbers are True.
- For example:
  - -11 → True
  - 0 → False
  - 99 → True

Relational Operators (! → NOT)

- Unary NOT operator converts True to False and False to True.

X	! X
True	False
False	True

- For example: a = !6 → 0

Relational Operators (&& → AND)

- Returns True if both conditions are True.

X	Y	X && Y
False	False	False
False	True	False
True	False	False
True	True	True

Relational Operators (&& → AND)

- First, left side of AND operator is evaluated. If left side of AND operator is false, evaluation stops.

- For example:
  - a = 4 && 0 → a = 0
  - b = 10 && -4 → b = 1

Relational Operators (|| → OR)

- Returns True if either of it's conditions are true.

X	Y	X || Y
False	False	False
False	True	True
True	False	True
True	True	True

Relational Operators (|| → OR)

- First, left side of OR operator is evaluated. If left side of OR operator is true, evaluation stops.
- For example:
  - a = 3 || 0 → a = 1
  - b = 0 || -30 → b = 1

Relational Operators

- The && operator has a higher precedence than ||.
- An expression containing && or || operators is evaluated only until truth or falsehood is known.
- This performance feature for the evaluation of logical AND and logical OR expressions is called short-circuit evaluation

Precedence of Operators

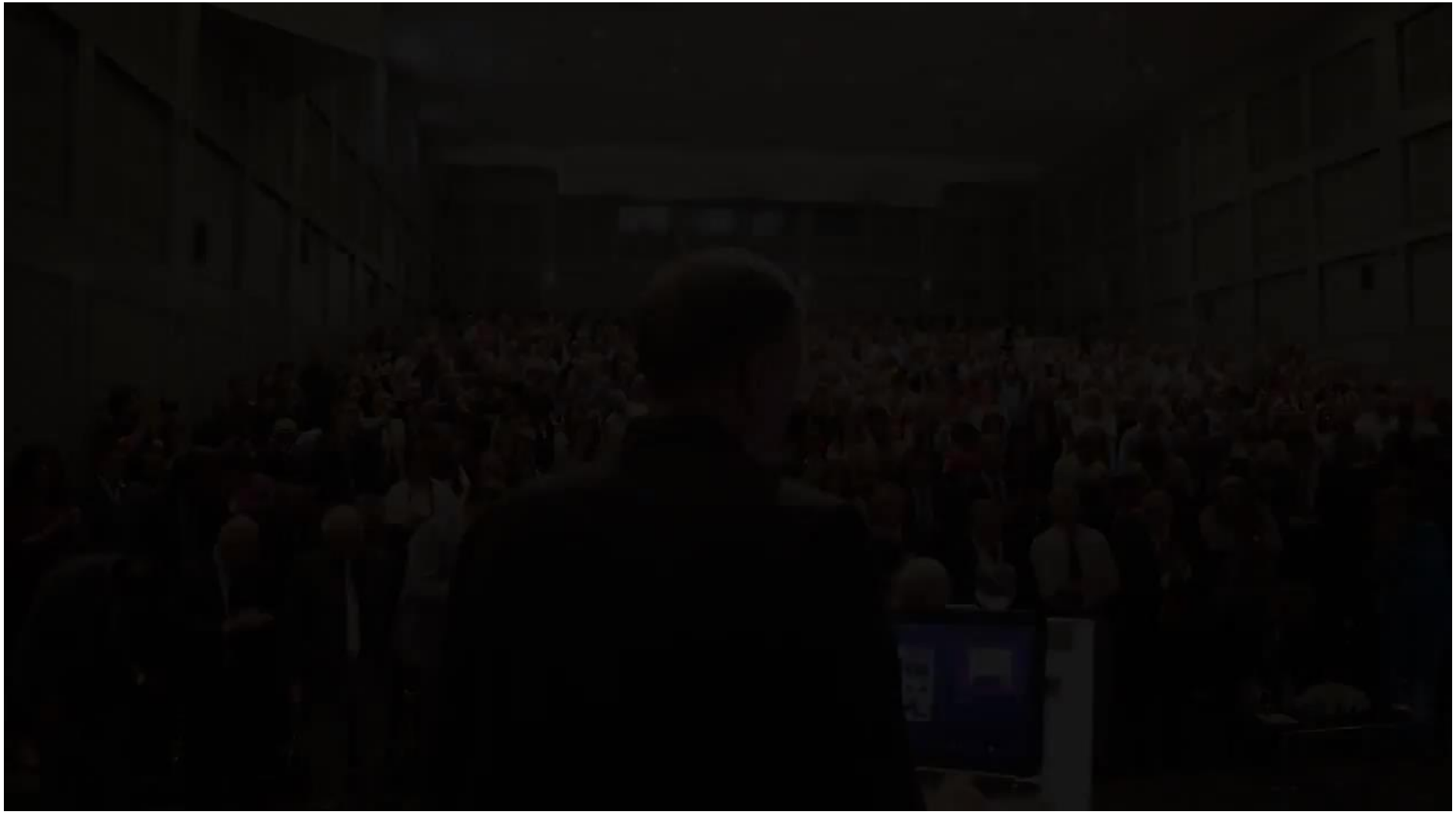
		HIGH PRECEDENCE	
( )	Left to right		Paranthesis
! ++ --	Right to left		Arithmetic op.
* / %	Left to right		
+ -	Left to right		
> >= < <=	Left to right		Relational op.
== !=	Left to right		
&&	Left to right		Logical op.
||	Left to right		
=	Right to left	LOW PRECEDENCE	Asignment op.

Notice that using parenthesis is the best way for not having mistake.

Example Operations in Operators

- Example1:
  - a= 15;
  - x = a >= 10 && a <= 20;
  - Here, x = 1
- Example2:
  - a= 20;
  - b= 10;
  - y = a + b >= 20 || a – b <= 10;
  - Here, y = 1

- Example3:
  - a= 5;
  - b= 0;
  - y = a || b && a && b
  - Here, y = 1



References

- Doç. Dr. Fahri Vatansever, “Algoritma Geliştirme ve Programlamaya Giriş”, Seçkin Yayıncılık, 12. Baskı, 2015
- J. G. Brookshear, “Computer Science: An Overview 10th Ed.”, Addison Wisley, 2009
- Kaan Aslan, “A’dan Z’ye C Klavuzu 8. Basım”, Pusula Yayıncılık, 2002
- Paul J. Deitel, “C How to Program”, Harvey Deitel.