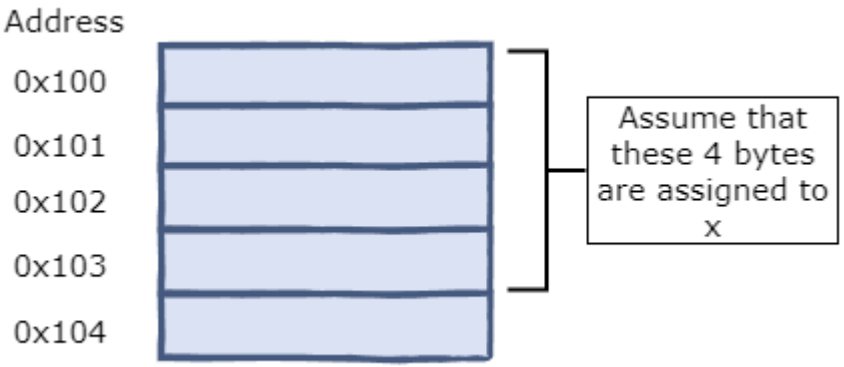
IS1201: Programming & Problem Solving

4. Operators

Viraj Welgama

sizeof()

- A computer's memory is a collection of byte-addressable chunks.



- sizeof() is a built-in function that is used to calculate the size (in bytes)that a data type occupies in the computer's memory.
  - Suppose that a variable x is of type integer and takes four bytes of the computer's memory, then sizeof(x) would return four.

sizeof()

- This function is a unary operator (i.e., it takes in one argument).
- This argument can be a;
  - Data type: The data type can be primitive (e.g., int, char, float) or user-defined (e.g., struct).
  - Expression

- IMORTANT NOTICE:
  - The result of the sizeof() function is machine-dependent since the sizes of data types in C varies from system to system.

sizeof()

```c
#include<stdio.h>

int main() {
    int x = 20;
    char y = 'a';
    //Using variable names as input
    printf("The size of int is: %d\n", sizeof(x));
    printf("The size of char is %d\n", sizeof(y));
    printf("The size of x + y is: %d\n",sizeof(x+y));
    //Using datatype as input
    printf("The size of float is: %d\n", sizeof(float));
    printf("The size of double is: %d\n", sizeof(double));
    return 0;
}
```

sizeof()

```c
#include<stdio.h>

int main() {
    int x = 20;
    char y = 'a';
    //Using variable names as input
    printf("The size of int is: %d\n", sizeof(x));
    printf("The size of char is %d\n", sizeof(y));
    printf("The size of x + y is: %d\n",sizeof(x+y));
    //Using datatype as input
    printf("The size of float is: %d\n", sizeof(float));
    printf("The size of double is: %d\n", sizeof(double));
    return 0;
}
```

```
The size of int is: 4
The size of char is 1
The size of x + y is: 4
The size of float is: 4
The size of double is: 8
```

Size qualifiers

- Size qualifiers alters the size of a basic type.
- There are two size qualifiers, long and short.
- For example:

```
long double x;
```

- The size of double is 8 bytes.
- However, when long keyword is used, that variable becomes 16 bytes.

```
double x;
long double lx;

printf("size of x: = %d\n", sizeof(x));
printf("size of lx: = %d", sizeof(lx));
```

Size qualifiers

- Size qualifiers alters the size of a basic type.
- There are two size qualifiers, long and short.

- The size of int is 4 bytes.
- However, when short keyword is used, that variable becomes 2 bytes.

```
int x;
short sx;

printf("size of x: = %d\n", sizeof(x));
printf("size of sx: = %d", sizeof(sx));
```

Problems

1. Write a C program to multiply two floating point numbers

2. Write a C program that computes the area of a disk.

3. Write a C Program to swap two numbers

Operators

- C programming has various types of operators to perform tasks including arithmetic, conditional and bitwise operations.
- Operators in C programming are;
  - Arithmetic Operators
  - Increment & Decrement Operators
  - Assignment Operators
  - Relational Operators
  - Logical Operators
  - Conditional Operators
  - Bitwise Operators
  - Special Operators

Arithmetic Operators

- An arithmetic operator performs mathematical operations such as addition, subtraction and multiplication on numerical values (constants and variables).

Operator	Meaning
+	Addition or unary plus
-	Subtraction or unary minus
*	Multifaction
/	Division
%	Remainder after division (modulo division)

Arithmetic Operators

```c
int main() {
    int a = 9, b = 4, c;
    c = a+b;
    printf("a+b = %d\n", c);
    c = a-b;
    printf("a-b = %d\n", c);
    c = a*b;
    printf("a*b = %d\n", c);
    c = a/b;
    printf("a/b = %d\n", c);
    c = a%b;
    printf("Remainder = %d\n", c);
}
```

Arithmetic Operators

```c
int main() {
    int a = 9, b = 4, c;
    c = a+b;
    printf("a+b = %d\n", c);
    c = a-b;
    printf("a-b = %d\n", c);
    c = a*b;
    printf("a*b = %d\n", c);
    c = a/b;
    printf("a/b = %d\n", c);
    c = a%b;
    printf("Remainder = %d\n", c);
}
```

```
a+b = 13
a-b = 5
a*b = 36
a/b = 2
Remainder = 1
```

Increment & Decrement Operators

- C programming has two increment and decrement operators to change the value of an operand (constant or variable) by 1.
  - Increment ++ increases the value by 1
  - Decrement -- decreases the value by 1
- These two operators are unary operators, meaning they only operate on a single operand.
- The operators ++ and -- can be used as prefix or postfix.

Increment & Decrement Operators

```c
int main() {
    int a = 7;
    float b = 5.5;
    printf("++a = %d\n", ++a);
    printf("--b = %.2f\n", --b);
    printf("a++ = %d\n", a++);
    printf("b-- = %.2f\n", b--);
    printf("Final Values: a = %d, b = %.2f\n", a, b);
}
```

Increment & Decrement Operators

```c
int main() {
    int a = 7;
    float b = 5.5;
    printf("++a = %d\n", ++a);
    printf("--b = %.2f\n", --b);
    printf("a++ = %d\n", a++);
    printf("b-- = %.2f\n", b--);
    printf("Final Values: a = %d, b = %.2f\n", a, b);
}
```

```
++a = 8
--b = 4.50
a++ = 8
b-- = 4.50
Final Values: a = 9, b = 3.50
```

Assignment Operators

- An assignment operator is used for assigning a value to a variable.
  - The most common assignment operator is =

Operator	Example	Same as…
=	a = b	a = b
+=	a += b	a = a + b
-=	a -= b	a = a - b
*=	a *= b	a = a * b
/=	a /= b	a = a / b
%=	a %= b	a = a % b

Assignment Operators

```c
int main() {
    int a = 5, b;
    b = a;
    printf("b = %d\n", b);
    b += a;
    printf("b = %d\n", b);
    b -= a;
    printf("b = %d\n", b);
    b *= a;
    printf("b = %d\n", b);
    b /= a;
    printf("b = %d\n", b);
    b %= a;
    printf("b = %d\n", b);
}
```

Assignment Operators

```c
int main() {
    int a = 5, b;
    b = a;
    printf("b = %d\n", b);
    b += a;
    printf("b = %d\n", b);
    b -= a;
    printf("b = %d\n", b);
    b *= a;
    printf("b = %d\n", b);
    b /= a;
    printf("b = %d\n", b);
    b %= a;
    printf("b = %d\n", b);
}
```

```
b = 5
b = 10
b = 5
b = 25
b = 5
b = 0
```

Relational Operators

- A relational operator checks the relationship between two operands
- If the relation is true, it returns 1; if the relation is false, it returns value 0
- Relational operators are used in decision making and loops.

Operator	Meaning	Example
==	Equal to	5 == 3 returns 0
>	Greater than	5 > 3 returns 1
<	Less than	5 < 3 returns 0
!=	Not equal to	5 != 3 returns 1
>=	Greater than or equal to	5 >= 3 returns 1
<=	Less than or equal to	5 <= 3 returns 0

Relational Operators

```c
int main() {

    int a = 5, b = 5, c = 10;

    printf("%d == %d = %d\n", a, b, a == b);
    printf("%d == %d = %d\n", a, c, a == c);

    printf("%d > %d = %d\n", a, b, a > b);
    printf("%d > %d = %d\n", a, c, a > c);

    printf("%d < %d = %d\n", a, b, a < b);
    printf("%d < %d = %d\n", a, c, a < c);

    printf("%d != %d = %d\n", a, b, a != b);
    printf("%d != %d = %d\n", a, c, a != c);

    printf("%d >= %d = %d\n", a, b, a >= b);
    printf("%d >= %d = %d\n", a, c, a >= c);

    printf("%d <= %d = %d\n", a, b, a <= b);
    printf("%d <= %d = %d\n", a, c, a <= c);

}
```

Relational Operators

```c
int main() {

    int a = 5, b = 5, c = 10;

    printf("%d == %d = %d\n", a, b, a == b);
    printf("%d == %d = %d\n", a, c, a == c);

    printf("%d > %d = %d\n", a, b, a > b);
    printf("%d > %d = %d\n", a, c, a > c);

    printf("%d < %d = %d\n", a, b, a < b);
    printf("%d < %d = %d\n", a, c, a < c);

    printf("%d != %d = %d\n", a, b, a != b);
    printf("%d != %d = %d\n", a, c, a != c);

    printf("%d >= %d = %d\n", a, b, a >= b);
    printf("%d >= %d = %d\n", a, c, a >= c);

    printf("%d <= %d = %d\n", a, b, a <= b);
    printf("%d <= %d = %d\n", a, c, a <= c);

}
```

```
5 == 5 = 1
5 == 10 = 0
5 > 5 = 0
5 > 10 = 0
5 < 5 = 0
5 < 10 = 1
5 != 5 = 0
5 != 10 = 1
5 >= 5 = 1
5 >= 10 = 0
5 <= 5 = 1
5 <= 10 = 1
```

Logical Operators

- An expression containing logical operator returns either 0 or 1 depending upon whether expression results true or false. Logical operators are commonly used in decision making in C programming.

Operator	Meaning	Example
&&	Logical AND, true only if all operands are true	If c = 5 & d = 2, then, expression ((c==5)&&(d==5)) equals to 0
||	Logical OR, true only if either operand is true	If c = 5 & d = 2, then, expression ((c==5)&&(d==5)) equals to 1
!	Logical NOT, true only if the operand is 0	If c = 5 then, expression !(c==5) equals to 0

Logical Operators

```c
int main() {

    int a = 5, b = 5, c = 10, result;

    result = (a == b) && (c > b);
    printf("(a == b) && (c > b) equals to %d\n", result);

    result = (a == b) && (c < b);
    printf("(a == b) && (c < b) equals to %d\n", result);

    result = (a == b) || (c > b);
    printf("(a == b) || (c > b) equals to %d\n", result);

    result = (a != b) || (c > b);
    printf("(a != b) || (c > b) equals to %d\n", result);

    result = !(a != b);
    printf("!(a != b) equals to %d\n", result);

    result = !(a == b);
    printf("!(a == b) equals to %d\n", result);
}
```

Logical Operators

```c
int main() {

    int a = 5, b = 5, c = 10, result;

    result = (a == b) && (c > b);
    printf("(a == b) && (c > b) equals to %d\n", result);

    result = (a == b) && (c < b);
    printf("(a == b) && (c < b) equals to %d\n", result);

    result = (a == b) || (c > b);
    printf("(a == b) || (c > b) equals to %d\n", result);

    result = (a != b) || (c > b);
    printf("(a != b) || (c > b) equals to %d\n", result);

    result = !(a != b);
    printf("!(a != b) equals to %d\n", result);

    result = !(a == b);
    printf("!(a == b) equals to %d\n", result);
}
```

```
(a == b) && (c > b) equals to 1
(a == b) && (c < b) equals to 0
(a == b) || (c > b) equals to 1
(a != b) || (c > b) equals to 1
!(a != b) equals to 1
!(a == b) equals to 0
```

Bitwise Operators

- During computation, mathematical operations like: addition, subtraction, addition and division are converted to bit-level which makes processing faster and saves power
- Bitwise operators are used in C programming to perform bit-level operations.

Operator	Meaning
&	Bitwise AND
|	Bitwise OR
^	Bitwise exclusive OR
~	Bitwise complement
<<	Shift left
>>	Shift right

Bitwise AND (&)

- The output of bitwise AND is 1 if the corresponding bits of two operands are 1.
  - If either bit of an operand is 0, the result of corresponding bit is evaluated to 0.

```
12  =  00001100 (in Binary)
25  =  00011001 (in Binary)

Bit operation of 12 and 25

     00001100
 &   00011001
 ----------

     00001000  =  8 (in Decimal)
```

Bitwise OR (|)

- The output of bitwise OR is 1 if at least one corresponding bit of two operands is 1

```
12  =  00001100 (in Binary)
25  =  00011001 (in Binary)

Bit operation of 12 and 25

    00001100
 |  00011001
 ----------

    00011101  =  29 (in Decimal)
```

Bitwise XOR (^)

- The result of bitwise XOR operator is 1 if the corresponding bits of two operands are opposite.

```
12  =  00001100 (in Binary)
25  =  00011001 (in Binary)

Bit operation of 12 and 25

    00001100
 ^  00011001
    --------

    00010101  =  21 (in Decimal)
```

Bitwise complement (~)

- Bitwise compliment operator is an unary operator (works on only one operand) which changes 1 to 0 and 0 to 1.

```
12  =  00001100 (in Binary)

Bitwise complement of 12

   ~ 00001100

     11110011  =  243 (in Decimal)
```

Bitwise Operators

```c
int main() {

    int a = 12, b = 25;

    printf("Bitwise AND = %d\n", a&b);
    printf("Bitwise OR = %d\n", a|b);
    printf("Bitwise XOR = %d\n", a^b);
    printf("Complement of a = %d\n", ~a);
    printf("Complement of b = %d\n", ~b);

}
```

Bitwise Operators

```c
int main() {

    int a = 12, b = 25;

    printf("Bitwise AND = %d\n", a&b);
    printf("Bitwise OR = %d\n", a|b);
    printf("Bitwise XOR = %d\n", a^b);
    printf("Complement of a = %d\n", ~a);
    printf("Complement of b = %d\n", ~b);

}
```

```
Bitwise AND = 8
Bitwise OR = 29
Bitwise XOR = 21
Complement of a = -13
Complement of b = -26
```

Shift Operators

- Right shift operator

```
212 = 11010100 (In binary)
212>>2 = 00110101 (In binary) [Right shift by two bits]
212>>7 = 00000001 (In binary)
212>>8 = 00000000
212>>0 = 11010100 (No Shift)
```

- Left shift operator

```
212 = 11010100 (In binary)
212<<1 = 110101000 (In binary) [Left shift by one bit]
212<<0 =11010100 (Shift by 0)
212<<4 = 110101000000 (In binary) =3392(In decimal)
```

Other Operators

- Comma Operator
  - Comma operators are used to link related expressions together.
  - For example:

```
int x = 10, y = 5, z;
```

- The sizeof operator
  - The *sizeof* is an unary operator which returns the size of data (constant, variables, array, structure etc).

- C Ternary Operator (?:)

Problems

1. The temperature is 35C; Write a program convert this temperature into Fahrenheit.
   - ºF =ºC * 1.8000 + 32.00

2. Write a C program that takes a number from the user and checks whether that number is either positive or negative or zero.

3. Write a C program to check a given character is Vowel or Consonant.