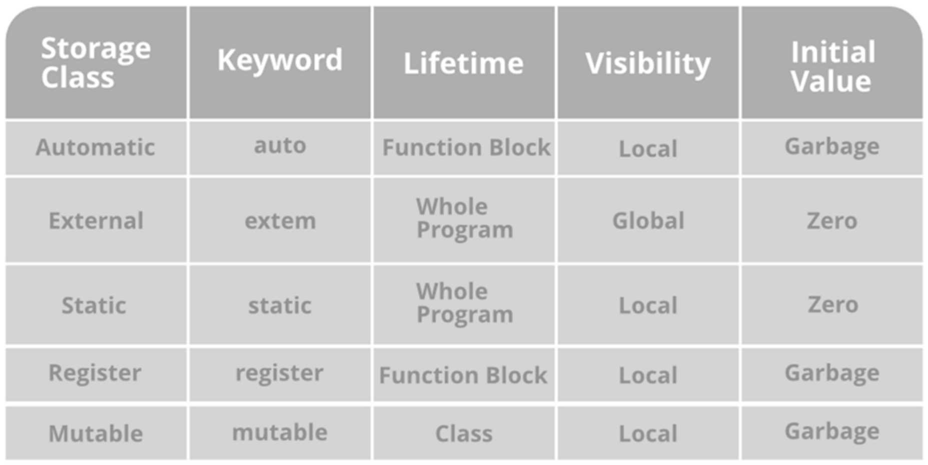
Variable Scope

- A scope is a region of the program. With the following three ways:
  - Inside a function or a block which is called local variables,
  - In the definition of function parameters which is called formal parameters.
  - Outside of all functions which is called global variables.
- When a local variable is defined, it is not initialized by the system, you must initialize it yourself. Global variables are initialized automatically by the system when you define them as follows:
  - Int/long as 0
  - Char as '\0'
  - float/double as 0
  - Pointer as NULL

37

Storage Classes

- auto
- register
- extern
- static
- mutable



Storage Class	Keyword	Lifetime	Visibility	Initial Value
Automatic	auto	Function Block	Local	Garbage
External	extem	Whole Program	Global	Zero
Static	static	Whole Program	Local	Zero
Register	register	Function Block	Local	Garbage
Mutable	mutable	Class	Local	Garbage

38

Local Vs Global

```
#include <iostream>
using namespace std;
 int main () {
   // Local variable declaration:
   int a, b;
   int c;
    // actual initialization
   a = 10;
   b = 20;
   c = a + b;
    cout << c;
    return 0;
}
```

```
#include <iostream>
using namespace std;
int g; // Global variable declaration
int main () {
  int a, b; // Local variable declaration:
  a = 10;
   b = 20;
   g = a + b;
   cout << g;
   return 0;
}
```

39

Operators

- Assignment operator (=)
- Arithmetic operators (+, -, *, /, %)
- Increment and decrement (++, --)
- Relational and comparison operators (==, !=, >, <, >=, <=)
- Logical operators (!, &&, ||)
- Conditional ternary operator (? :)
- Comma operator (,)
- Bitwise operators (&, |, ^, ~, <<, >>)
- Explicit type casting operator
- sizeof

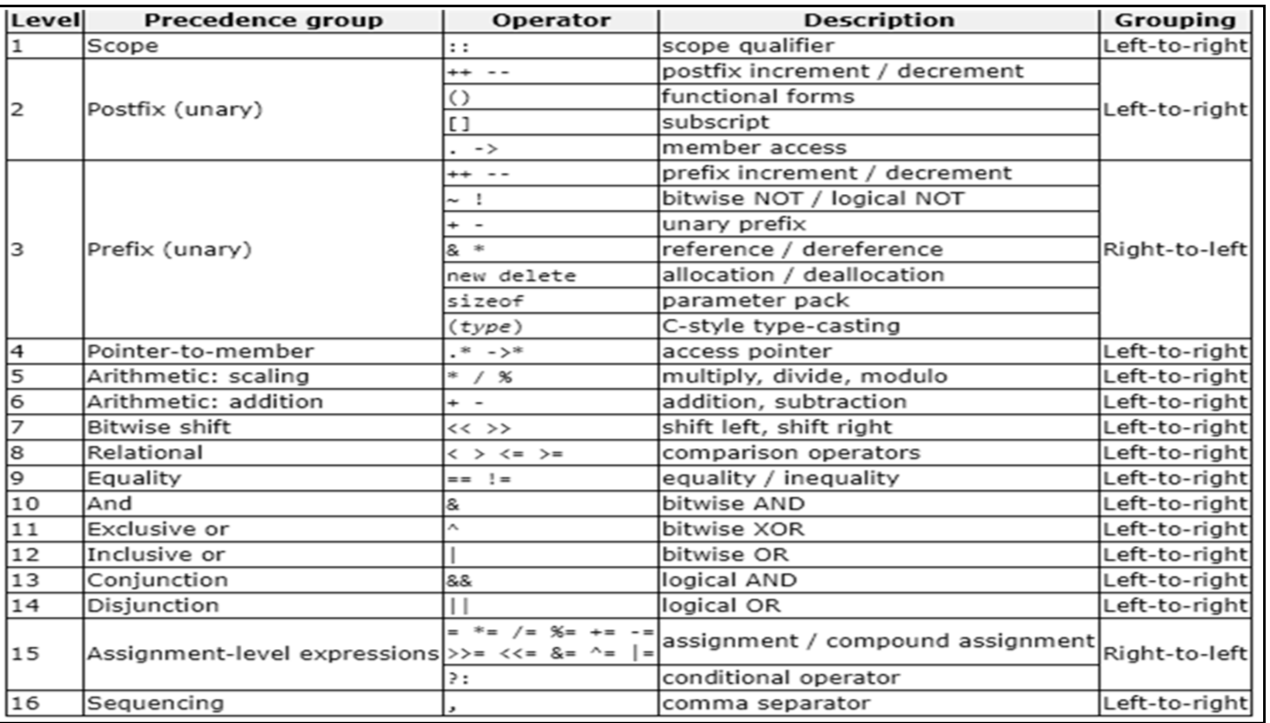
40

Auto

The auto keyword provides type inference capabilities

```
#include <iostream>
using namespace std;
 void autoStorageClass()
{
   cout << "Demonstrating auto class\n";
   auto a = 32;
   auto b = 3.2;
   auto c = "GeeksforGeeks";
   auto d = 'G';
 // printing the auto variables
   cout << a << " \n"<< b << " \n"<< c << " \n"<< d << " 
\n";
}

int main()
{

   // To demonstrate auto Storage Class
   autoStorageClass();

   return 0;
}
```

41



Level	Precedence group	Operator	Description	Grouping
1	Scope	`::`	scope qualifier	Left-to-right
2	Postfix (unary)	`++ --`	postfix increment / decrement	Left-to-right
		`()`	functional forms	
		`[]`	subscript	
		`. ->`	member access	
3	Prefix (unary)	`++ --`	prefix increment / decrement	Right-to-left
		`~ !`	bitwise NOT / logical NOT	
		`+ -`	unary prefix	
		`& *`	reference / dereference	
		`new delete`	allocation / deallocation	
		`sizeof`	parameter pack	
		`(type)`	C-style type-casting	
4	Pointer-to-member	`.* ->*`	access pointer	Left-to-right
5	Arithmetic: scaling	`* / %`	multiply, divide, modulo	Left-to-right
6	Arithmetic: addition	`+ -`	addition, subtraction	Left-to-right
7	Bitwise shift	`<< >>`	shift left, shift right	Left-to-right
8	Relational	`< > <= >=`	comparison operators	Left-to-right
9	Equality	`== !=`	equality / inequality	Left-to-right
10	And	`&`	bitwise AND	Left-to-right
11	Exclusive or	`^`	bitwise XOR	Left-to-right
12	Inclusive or	`|`	bitwise OR	Left-to-right
13	Conjunction	`&&`	logical AND	Left-to-right
14	Disjunction	`||`	logical OR	Left-to-right
15	Assignment-level expressions	`= *= /= %= += -= >>= <<= &= ^= |=`	assignment / compound assignment	Right-to-left
		`?:`	conditional operator	
16	Sequencing	`,`	comma separator	Left-to-right

42

Assignment Operator

```
#include <iostream>
using namespace std;
int main ()
{
  int a, b;         // a:?,  b:?
  a = 10;           // a:10, b:?
  b = 4;            // a:10, b:4
  a = b;            // a:4,  b:4
  b = 7;            // a:4,  b:7
  cout << "a:";
  cout << a;
  cout << " b:";
  cout << b;
}
```

```
y = 2 + (x = 5);
```

7

43

Arithmetic operators (+, -, *, /, %)

- Mathematical Operations
- x = 11 % 3;
- 2

44

Lets Code

Write a program to input time in seconds and convert them to hours, minutes and seconds

Compound assignment (+=, -=, *=, /=, %=, >>=, <<=, &=, ^=, |=)

expression	equivalent to...
y += x;	y = y + x;
x -= 5;	x = x - 5;
x /= y;	x = x / y;
price *= units + 1;	price = price * (units+1);

Compound Operator(example)

```
#include <iostream>
using namespace std;

int main ()
{
  int a, b=3;
  a = b;
  a+=2;          // equivalent to a=a+2
  cout << a;
}
```

Increment and decrement (++, --)

Example 1	Example 2

x = 3;
y = ++x;
// x contains 4, y contains 4
	x = 3;
	y = x++;
	// x contains 4, y contains 3

Relational and comparison operators (==, !=, >, <, >=, <=)

```
(7 == 5)    // false
(5 > 4)     // true
(3 != 2)    // true
(6 >= 6)    // true
(5 < 5)     // false
```

Now, Suppose that a=2, b=3 and c=6, then:

```
(a == 5)    // false, since a is not equal to 5
(a*b >= c)  // true, since (2*3 >= 6) is true
(b+4 > a*c) // false, since (3+4 > 2*6) is false
((b=2) == a) // true
```

49

Logical operators (!, &&, ||)

&& OPERATOR (and)		
a	b	a && b
true	true	true
true	false	false
false	true	false
false	false	false

|| OPERATOR (or)		
a	b	a || b
true	true	true
true	false	true
false	true	true
false	false	false

!(5 == 5)
// evaluates to false because the expression at its right (5 == 5) is true

!(6 <= 4)
// evaluates to true because (6 <= 4) would be false

!true // evaluates to false

!false // evaluates to true

50

Conditional ternary operator (? :)

```
#include <iostream>
using namespace std;
int main ()
{
  int a,b,c;
  a=2;
  b=7;
  c = (a>b) ? a : b;
 cout << c << '\n';
}
```

51

Comma operator (,)

- used to separate two or more expressions that are included where only one expression is expected.
- When the set of expressions has to be evaluated for a value, only the right-most expression is considered.
- a = (b=3, b+2);

52

Bitwise operators (&, |, ^, ~, <<, >>)

```
#include <iostream>
using namespace std;
int main()
{
        int a=1,b=5;
        cout<<(a&b)<<endl;
        cout<<(a|b)<<endl;
        cout<<(a^b)<<endl;
        cout<<(a>>2)<<endl;
        cout<<(a<<2)<<endl;
        cout<<(~a)<<endl;
        return 0;
}
```

53

Operator Precedence

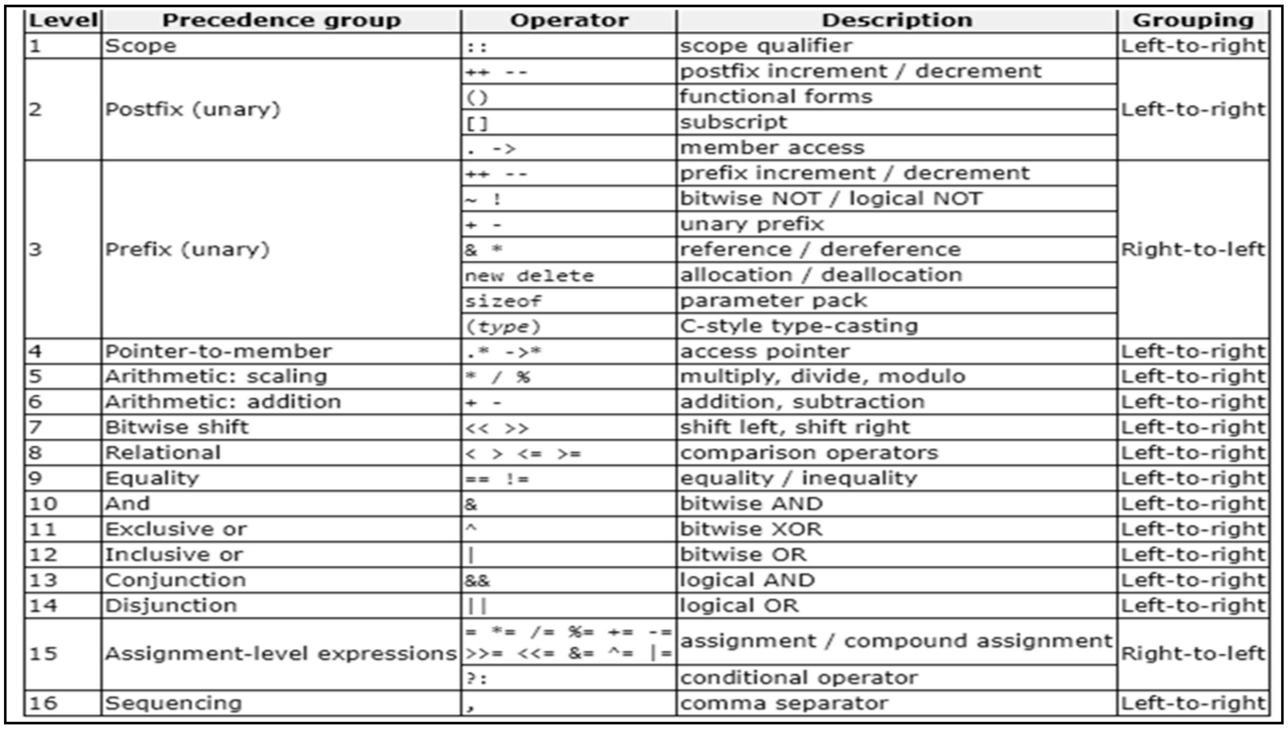
```
#include <iostream>
using namespace std;

int main() {
        int num1 = 5 - 17 * 6;
        int num2 = 5 - (17 * 6);
        int num3 = (5 - 17) * 6;
        cout << "num1 = " << num1 << endl;
        cout << "num2 = " << num2 << endl;
        cout << "num3 = " << num3 << endl;
        return 0;
}
```

54

Operator Precedence

```
#include <iostream>
using namespace std;
int main() {
        int a = 1;
        int b = 4;
        b += a -= 6;
        cout << "a = " << a << endl; ;
        cout << "b = " << b;
}
```

55



Level	Precedence group	Operator	Description	Grouping
1	Scope	`::`	scope qualifier	Left-to-right
2	Postfix (unary)	`++ --`	postfix increment / decrement	Left-to-right
		`()`	functional forms	
		`[]`	subscript	
		`. ->`	member access	
3	Prefix (unary)	`++ --`	prefix increment / decrement	Right-to-left
		`~ !`	bitwise NOT / logical NOT	
		`+ -`	unary prefix	
		`& *`	reference / dereference	
		`new delete`	allocation / deallocation	
		`sizeof`	parameter pack	
		`(type)`	C-style type-casting	
4	Pointer-to-member	`.* ->*`	access pointer	Left-to-right
5	Arithmetic: scaling	`* / %`	multiply, divide, modulo	Left-to-right
6	Arithmetic: addition	`+ -`	addition, subtraction	Left-to-right
7	Bitwise shift	`<< >>`	shift left, shift right	Left-to-right
8	Relational	`< > <= >=`	comparison operators	Left-to-right
9	Equality	`== !=`	equality / inequality	Left-to-right
10	And	`&`	bitwise AND	Left-to-right
11	Exclusive or	`^`	bitwise XOR	Left-to-right
12	Inclusive or	`|`	bitwise OR	Left-to-right
13	Conjunction	`&&`	logical AND	Left-to-right
14	Disjunction	`||`	logical OR	Left-to-right
15	Assignment-level expressions	`= *= /= %= += -= >>= <<= &= ^= |=`	assignment / compound assignment	Right-to-left
		`?:`	conditional operator	
16	Sequencing	`,`	comma separator	Left-to-right

56

Lets evaluate Expressions

x = 10 -20+ 22 / 2

x=10 – 20+11

x=-10+11

x=1

57

Lets evaluate Expressions

x = 10 == 22 != 2

x=0 != 2

x=1

58

Lets evaluate Expressions

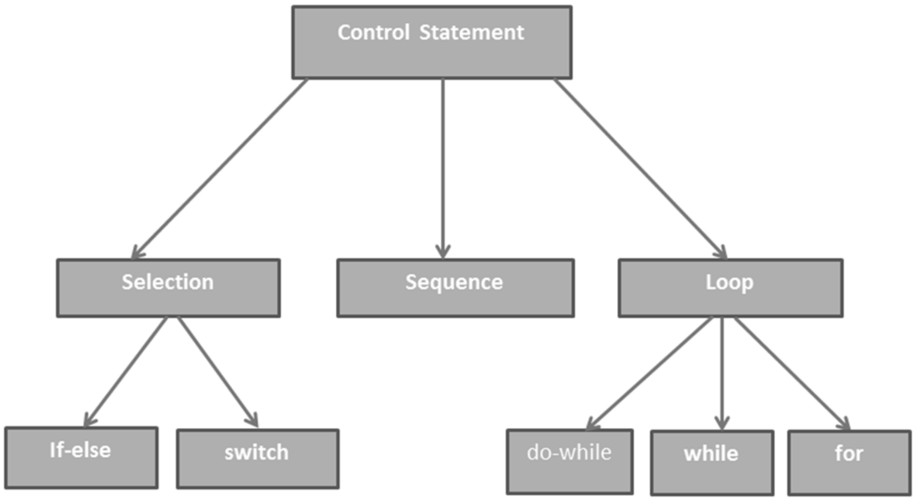
x = 10 == (22 != 2)

x=10 == 1

x=0

59

Control Structures



60