```
/*
 * C++ Programming Notes
 * Ravi Kumar Reddy K
 * github.com/ravikumark815
*/
```

Preset:

- Invented by Bjarne Stroustrup in 1979
- Middle Level Language
- Versions: C++ 14, C++11, C++99

Hello World:

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

int imGlobal = 0;
const double PI = 3.141;

int main(int argc, char**argv) {
  cout << "Hello World\n";
  return 0;
}
```

- Namespaces
- main: Start executing from here
- Cout allows us to output information to console
- “<<” Stream insertion operator: Takes string on the right to cout stream
- “endl” Issue newline and force write to console
- argc: No of arguments passed to main
- argv: Array of pointers to strings in the arg vector
- int: Return an integer when done executing
- imGlobal: Global variable and accessible everywhere else.
- const double PI: Global variable whose value cannot be changed anywhere else

Comments:

```
/*
Multi
Line
Comment
*/
// Single Line Comment
```

Common Header files:

- #include <cstdlib> // Sorting, Searching, import c libraries, rand, memmgmt, and other general-purpose functions
- #include <iostream> // Read and Write data
- #include <string> // Work with strings
- #include <limits> // Min and max values
- #include <vector> // Work with vectors
- #include <sstream> // Work with string streams
- #include <numeric> // Work with sequences of values
- #include <ctime> // Work with time
- #include <cmath> //Common math functions

Data Types:

Type	Typical Bit Width	Typical Range
char	1byte	-127 to 127 or 0 to 255
unsigned char	1byte	0 to 255
signed char	1byte	-127 to 127
int	4bytes	-2147483648 to 2147483647
unsigned int	4bytes	0 to 4294967295
signed int	4bytes	-2147483648 to 2147483647
short int	2bytes	-32768 to 32767
unsigned short int	2bytes	0 to 65,535
signed short int	2bytes	-32768 to 32767
long int	8bytes	-9223372036854775808 to 9223372036854775807
signed long int	8bytes	same as long int
unsigned long int	8bytes	0 to 18446744073709551615
long long int	8bytes	-(2^63) to (2^63)-1
unsigned long long int	8bytes	0 to 18,446,744,073,709,551,615
float	4bytes	
double	8bytes	
long double	12bytes	
wchar_t	2 or 4 bytes	1 wide character

Data Type	Initializer
int	0
char	'\0'
float	0
double	0
pointer	NULL

Variables:

- Definition: type variable_list = value;
- Ex: int i,j,k=10; char c,ch;

Type Qualifiers:

Sr.No	Qualifier & Meaning
1	**const** Objects of type **const** cannot be changed by your program during execution.
2	**volatile** The modifier **volatile** tells the compiler that a variable's value may be changed in ways not explicitly specified by the program.
3	**restrict** A pointer qualified by **restrict** is initially the only means by which the object it points to can be accessed. Only C99 adds a new type qualifier called restrict.

Storage Qualifiers:

Storage Class	Keyword	Lifetime	Visibility	Initial Value
Automatic	auto	Function Block	Local	Garbage
External	extern	Whole Program	Global	Zero
Static	static	Whole Program	Local	Zero
Register	register	Function Block	Local	Garbage
Mutable	mutable	Class	Local	Garbage
Thread Local	thread_local	whole thread	Local or Global	Garbage

Input and Output:

- cout << “Min int” << numeric_limits<int>::min();
- cout << “Max short int” << numeric_limits<short int>::max();
- printf(“Sum = %.7f\n”), (1.1111111+1.1111111)); // To print formatted output of float upto 7 decimal places
- cout << “int Byte:” << sizeof(int) << endl;
- printf(“%c %d %5d %.3f %s\n”, ‘A’, 10, 5, 3.1234, “Hi); // O/p: A 10 5 3.123 Hi //Right justify
- cin >> num_str; //to take in input for num1
- int num1 = stoi(num_str) //To convert num1 from string to int;
- bool res=true; cout.setf(ios::boolalpha); cout << res << endl; // To print booleans

Escape sequence	Meaning
\\	\ character
\'	' character
\"	" character
\?	? character
\a	Alert or bell
\b	Backspace
\f	Form feed
\n	Newline
\r	Carriage return
\t	Horizontal tab
\v	Vertical tab
\ooo	Octal number of one to three digits
\xhh . . .	Hexadecimal number of one or more digits

Operators:

Arithmetic Operators:

Operator	Description	Example
+	Adds two operands	A + B will give 30
-	Subtracts second operand from the first	A - B will give -10
*	Multiplies both operands	A * B will give 200
/	Divides numerator by de-numerator	B / A will give 2
%	Modulus Operator and remainder of after an integer division	B % A will give 0
++	**Increment operator**, increases integer value by one	A++ will give 11
--	**Decrement operator**, decreases integer value by one	A-- will give 9

Logical Operators:

Operator	Description	Example
&&	Called Logical AND operator. If both the operands are non-zero, then condition becomes true.	(A && B) is false.
||	Called Logical OR Operator. If any of the two operands is non-zero, then condition becomes true.	(A || B) is true.
!	Called Logical NOT Operator. Use to reverses the logical state of its operand. If a condition is true, then Logical NOT operator will make false.	!(A && B) is true.

Relational Operators:

Operator	Description	Example
==	Checks if the values of two operands are equal or not, if yes then condition becomes true.	(A == B) is not true.
!=	Checks if the values of two operands are equal or not, if values are not equal then condition becomes true.	(A != B) is true.
>	Checks if the value of left operand is greater than the value of right operand, if yes then condition becomes true.	(A > B) is not true.
<	Checks if the value of left operand is less than the value of right operand, if yes then condition becomes true.	(A < B) is true.
>=	Checks if the value of left operand is greater than or equal to the value of right operand, if yes then condition becomes true.	(A >= B) is not true.
<=	Checks if the value of left operand is less than or equal to the value of right operand, if yes then condition becomes true.	(A <= B) is true.

Bitwise Operators:

Operator	Description	Example
&	Binary AND Operator copies a bit to the result if it exists in both operands.	(A & B) will give 12 which is 0000 1100
|	Binary OR Operator copies a bit if it exists in either operand.	(A | B) will give 61 which is 0011 1101
^	Binary XOR Operator copies the bit if it is set in one operand but not both.	(A ^ B) will give 49 which is 0011 0001
~	Binary Ones Complement Operator is unary and has the effect of 'flipping' bits.	(~A) will give -61 which is 1100 0011 in 2's complement form due to a signed binary number.
<<	Binary Left Shift Operator. The left operands value is moved left by the number of bits specified by the right operand.	A << 2 will give 240 which is 1111 0000
>>	Binary Right Shift Operator. The left operands value is moved right by the number of bits specified by the right operand.	A >> 2 will give 15 which is 0000 1111

Assignment Operators:

Operator	Description	Example
=	Simple assignment operator, Assigns values from right side operands to left side operand.	C = A + B will assign value of A + B into C
+=	Add AND assignment operator, It adds right operand to the left operand and assign the result to left operand.	C += A is equivalent to C = C + A
-=	Subtract AND assignment operator, It subtracts right operand from the left operand and assign the result to left operand.	C -= A is equivalent to C = C - A
*=	Multiply AND assignment operator, It multiplies right operand with the left operand and assign the result to left operand.	C *= A is equivalent to C = C * A
/=	Divide AND assignment operator, It divides left operand with the right operand and assign the result to left operand.	C /= A is equivalent to C = C / A
%=	Modulus AND assignment operator, It takes modulus using two operands and assign the result to left operand.	C %= A is equivalent to C = C % A
<<=	Left shift AND assignment operator.	C <<= 2 is same as C = C << 2
>>=	Right shift AND assignment operator.	C >>= 2 is same as C = C >> 2
&=	Bitwise AND assignment operator.	C &= 2 is same as C = C & 2
^=	Bitwise exclusive OR and assignment operator.	C ^= 2 is same as C = C ^ 2
|=	Bitwise inclusive OR and assignment operator.	C |= 2 is same as C = C | 2

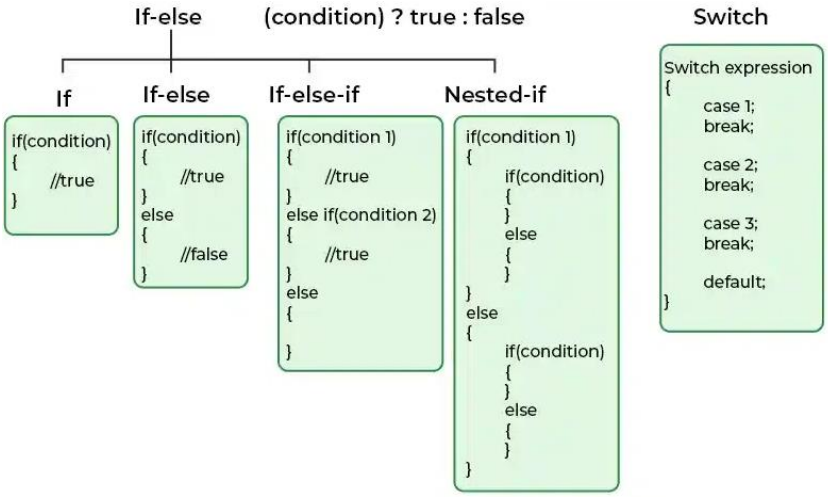
Misc Operators:

Sr.No	Operator & Description
1	**sizeof** **sizeof operator** returns the size of a variable. For example, sizeof(a), where 'a' is integer, and will return 4.
2	**Condition ? X : Y** **Conditional operator (?)**. If Condition is true then it returns value of X otherwise returns value of Y.
3	**,** **Comma operator** causes a sequence of operations to be performed. The value of the entire comma expression is the value of the last expression of the comma-separated list.
4	**. (dot) and -> (arrow)** **Member operators** are used to reference individual members of classes, structures, and unions.
5	**Cast** **Casting operators** convert one data type to another. For example, int(2.2000) would return 2.
6	**&** **Pointer operator &** returns the address of a variable. For example &a; will give actual address of the variable.
7	***** **Pointer operator *** is pointer to a variable. For example *var; will pointer to a variable var.

Precedence, Associativity:

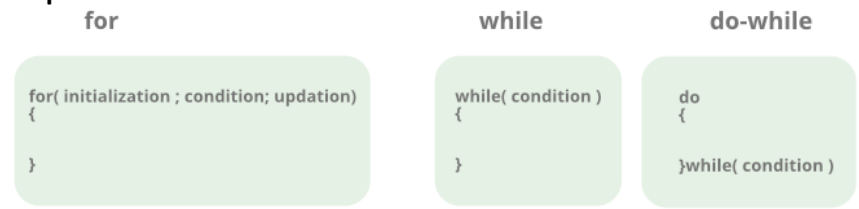
Precedence	Operator	Description	Associativity
1	: :	Scope resolution	Left to right
2	a++ a— type() type{} a() a[] . ->	Postfix increment and decrement Function cast Function call Subscript Member access	Left to right
3	++a - -a +a -a ! ~ (type) *a &a sizeof co_wait new new[ ] delete delete[ ]	Prefix increment and decrement Unary plus and minus Logical and bitwise NOT C-Style cast Dereference Address of Size-of Await expression Dynamic memory allocation Dynamic memory deallocation	Right to left
4	.* ->*	Pointer to member	Left to right
5	a*b a/b a%b	Multiplication, division, remainder	
6	a+b a-b	Addition , subraction	
7	<< >>	Bitwise left and right shift operators	
8	<= >	Three way comparision	
9	< <= > >=	Relational operators	
10	== !=	Equality and not equality check operators	
11	&	Bitwise AND	
12	^	Bitwise XOR	
13	|	Bitwise OR	
14	&&	Logical AND	
15	||	Logical OR	
16	a ? b:c throw co_yield = += -= *= /= %= <<= >>= &= ^= |=	Ternary conditional operator throw operator yield-expression Direct assignment Compound assignment by sum, difference Compound assignment by product,quotient,remainder Compound assignment by bitwise left and right shift Compound assignment bY bitwise AND, XOR, OR	Right to left
17	,	comma	Left to right

Conditional Statements:



Sr.No	Statement & Description
1	**if statement** An 'if' statement consists of a boolean expression followed by one or more statements.
2	**if...else statement** An 'if' statement can be followed by an optional 'else' statement, which executes when the boolean expression is false.
3	**switch statement** A 'switch' statement allows a variable to be tested for equality against a list of values.
4	**nested if statements** You can use one 'if' or 'else if' statement inside another 'if' or 'else if' statement(s).
5	**nested switch statements** You can use one 'switch' statement inside another 'switch' statement(s).

Loops:



Sr.No	Loop Type & Description
1	**while loop** Repeats a statement or group of statements while a given condition is true. It tests the condition before executing the loop body.
2	**for loop** Execute a sequence of statements multiple times and abbreviates the code that manages the loop variable.
3	**do...while loop** Like a 'while' statement, except that it tests the condition at the end of the loop body.
4	**nested loops** You can use one or more loop inside any another 'while', 'for' or 'do..while' loop.

Sr.No	Control Statement & Description
1	**break statement** Terminates the **loop** or **switch** statement and transfers execution to the statement immediately following the loop or switch.
2	**continue statement** Causes the loop to skip the remainder of its body and immediately retest its condition prior to reiterating.
3	**goto statement** Transfers control to the labeled statement. Though it is not advised to use goto statement in your program.

```
while (i <= 20){
      // If a value is even don't print it
      if((i % 2) == 0){
         i += 1;

         // Continue skips the rest of the code
         // and jumps back to the beginning
         // of the loop
         continue;
      }

      // Break stops execution of the loop and jumps
      // to the line after the loops closing }
      if(i == 15) break;

      cout << i << "\n";

      // Increment i so the loop eventually ends
      i += 1;
   }

// An abbreviated for loop
   int arr3[] = {1,2,3};
   for(auto x: arr3) cout << x << endl;

// Do while loops are guaranteed to execute at
   // least once
   // We'll create a secret number guessing game

   // We need to seed the random number generator
   // time() returns the number of seconds
   // since 1, 1, 1970
   // Include <ctime>
   srand(time(NULL));

   // Generate a random number up to 10
   int secretNum = rand() % 11;
   int guess = 0;
   do{
      cout << "Guess the Number : ";
      cin >> guess;
      if(guess > secretNum) cout << "To Big\n";
      if(guess < secretNum) cout << "To Small\n";
   } while(secretNum != guess);

   cout << "You guessed it" << endl;
```

Arrays:

```
void main(int argc, char**argv) {
            int array1 [10] = {1};       // Size
            int array2 [] = {1,2,3};     // Size for this would automatically be 3
            int array3 [5] = {8,9};      //
            cout << “First val: ” << array1[0] << endl;
```

```
        array1[0] = 7;
        int array4[2][3][3] = { {{1,2}, {3,4}}, {{5,6}, {7,8}} };    // Multidimensional arrays
        cout << array4[0][1][1] <<endl //prints 4
        return 0;
}
```

- Size once defined cannot be changed.

Vectors:

```
// ---------- VECTORS ----------
   // Vectors are used when you don't know how big the array
   // should be
   vector<int> vNums(2);

   // Add values
   vNums[0] = 1;
   vNums[1] = 2;

   // Add another to the end
   vNums.push_back(3);

   // Get vector size
   cout << "Vector Size : " << vNums.size() << endl;
```

String Streams:

```
// A stringstream object receives strings separated
   // by a space and then spits them out 1 by 1
   vector<string> words;
   stringstream ss("Some Random Words");
   string word;

   // A while loop will execute as long as there are
   // more words
   while(getline(ss, word, ' ')){
      words.push_back(word);
   }
// Cycle through each index in the vector using
   // a for loop
   for(int i = 0; i < words.size(); ++i){
      cout << words[i] << endl;
   }
```

Strings:

```
// A C++ string is a series of characters that
  // can be changed
  string str1 = "I'm a string";

  // Get the 1st character
  cout << "1st : " << str1[0] << endl;

  // Get the last character
  cout << "Last : " << str1.back() << endl;
```

```
   // Get the string length
   cout << "Length : " << str1.length() << endl;

   // Copy a string to another
   string str2 = str1;

   // Copy a string after the 1st 4 characters
   string str3(str2, 4);

   // Combine strings
   string str4 = str1 + " and your not";

   // Append to the end of a string
   str4.append("!");

   // Erase characters from a string from 1 index
   // to another
   str4.erase(12, str4.length() - 1);
   cout << "New String : " << str4 << endl;

   // find() returns index where pattern is found
   // or npos (End of String)
   if(str4.find("string") != string::npos)
      cout << "String Index : " <<
           str4.find("string") << endl;
// O/p: String Index: 6

   // substr(x, y) returns a substring starting at
   // index x with a length of y
   cout << "Substring : " <<
        str4.substr(6,6) << endl;
//O/p: Substring: string

   // Convert int to string
   string strNum = to_string(1+2);
   cout << "I'm a String : " << strNum << "\n";
//O/p: I'm a String: 3
```

Character functions

```
   char letterZ = 'z';
   char num5 = '5';
   char aSpace = ' ';
   cout << "Is z a letter or number " <<
        isalnum(letterZ) << endl;
   cout << "Is z a letter " <<
        isalpha(letterZ) << endl;
   cout << "Is 3 a number " <<
        isdigit(num5) << endl;
   cout << "Is space a space " <<
        isspace(aSpace) << endl;
```

Math Functions:

```
   cout << "abs(-10) = " << abs(-10) << endl;
```

```
    cout << "max(5, 4) = " << max(5, 4) << endl;
    cout << "min(5, 4) = " << min(5, 4) << endl;
    cout << "fmax(5.3, 4.3) = " << fmax(5.3, 4.3) << endl;
    cout << "fmin(5.3, 4.3) = " << fmin(5.3, 4.3) << endl;
    cout << "ceil(10.45) = " << ceil(10.45) << endl;
    cout << "floor(10.45) = " << floor(10.45) << endl;
    cout << "round(10.45) = " << round(10.45) << endl;
    cout << "pow(2,3) = " << pow(2,3) << endl;
    cout << "sqrt(100) = " << sqrt(100) << endl;
    cout << "cbrt(1000) = " << cbrt(1000) << endl;
// e ^ x
    cout << "exp(1) = " << exp(1) << endl;

    // 2 ^ x
    cout << "exp2(1) = " << exp2(1) << endl;

    // e * e * e ~= 20 so log(20.079) ~= 3
    cout << "log(20.079) = " << log(20.079) << endl;

    // 2 * 2 * 2 = 8
    cout << "log2(8) = " << log2(8) << endl;

    // Hypotenuse : SQRT(A^2 + B^2)
    cout << "hypot(2,3) = " << hypot(2,3) << endl;

// Also sin, cos, tan, asin, acos, atan, atan2,
    // sinh, cosh, tanh, asinh, acosh, atanh
```

Functions: