HASAN KALYONCU ÜNİVERSİTESİ

# Session 2: C++ Introduction

Data Structures and Algorithm 1 - Lab

Yahya Tawil
1 Oct 2021

# C++ Introduction

- The main references for our revision of C++ is:
  ***The C++ Language, Libraries, Tools, and Other Topics***
  http://www.ece.uvic.ca/~mdadams/cppbook .

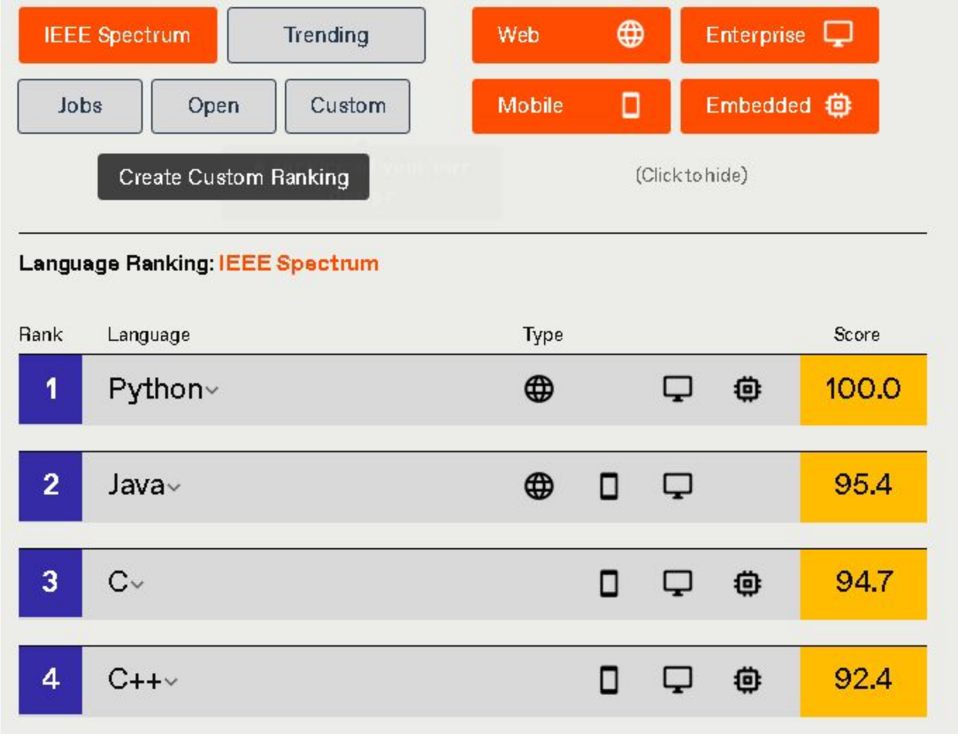
- Originally C with Classes, renamed as C++ in 1983 (superset of C).

- Supports object-oriented.

- Maintains efficiency of C.

- Where C++ is used: Desktop application software, device drivers, embedded software, high-performance server and client applications, video games and native code for Android applications.

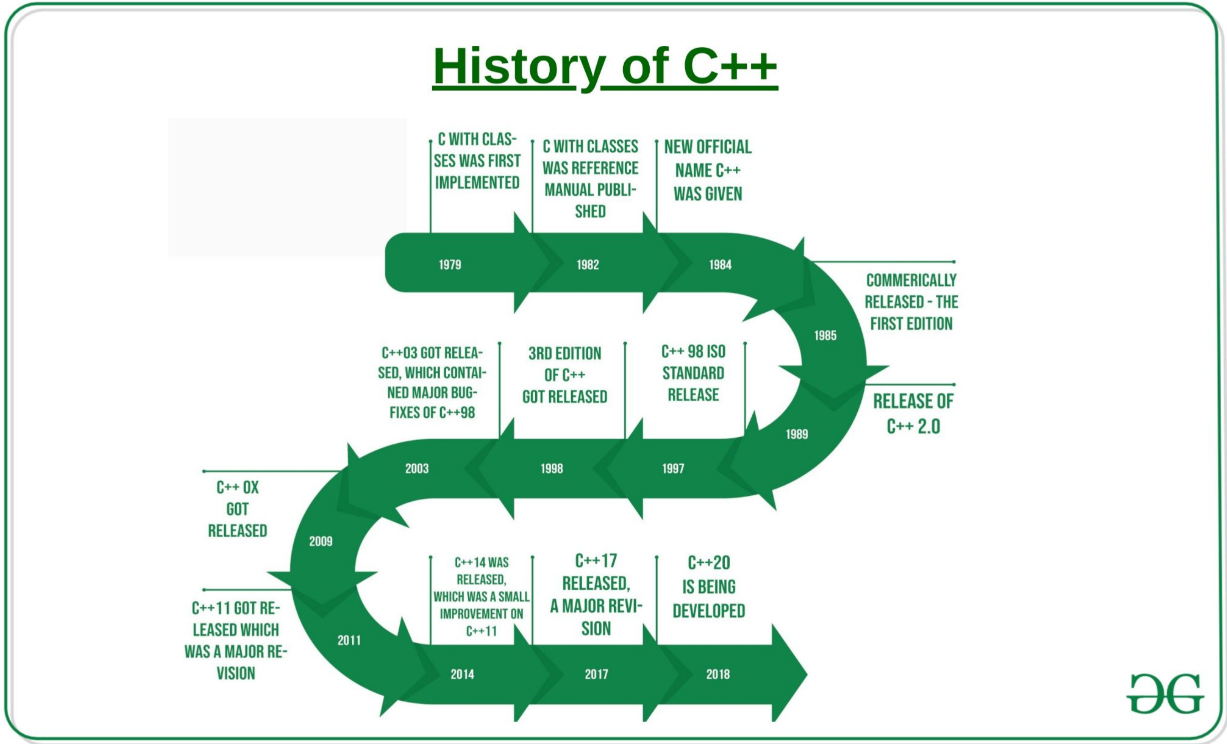
# spectrum.ieee.org/top-programming-languages 2021 programming languages Survey



Language Ranking: IEEE Spectrum

| Rank | Language | Type | | | | Score |
|---|---|---|---|---|---|---|
| 1 | Python | 🌐 | | 🖥 | ⚙ | 100.0 |
| 2 | Java | 🌐 | 📱 | 🖥 | | 95.4 |
| 3 | C | | 📱 | 🖥 | ⚙ | 94.7 |
| 4 | C++ | | 📱 | 🖥 | ⚙ | 92.4 |

# C++ History



## History of C++

1979

C WITH CLASSES WAS FIRST IMPLEMENTED

1982

C WITH CLASSES WAS REFERENCE MANUAL PUBLISHED

1984

NEW OFFICIAL NAME C++ WAS GIVEN

1985

COMMERICALLY RELEASED - THE FIRST EDITION

1989

RELEASE OF C++ 2.0

1997

C++ 98 ISO STANDARD RELEASE

1998

3RD EDITION OF C++ GOT RELEASED

2003

C++03 GOT RELEASED, WHICH CONTAINED MAJOR BUG-FIXES OF C++98

2009

C++ 0X GOT RELEASED

2011

C++11 GOT RELEASED WHICH WAS A MAJOR REVISION

2014

C++14 WAS RELEASED, WHICH WAS A SMALL IMPROVEMENT ON C++11

2017

C++17 RELEASED, A MAJOR REVISION

2018

C++20 IS BEING DEVELOPED

# Hello World in C++

Code

```
#include <iostream>


int main(){
    std::cout<<"Hello World\r\n";
    return 0;
}
```

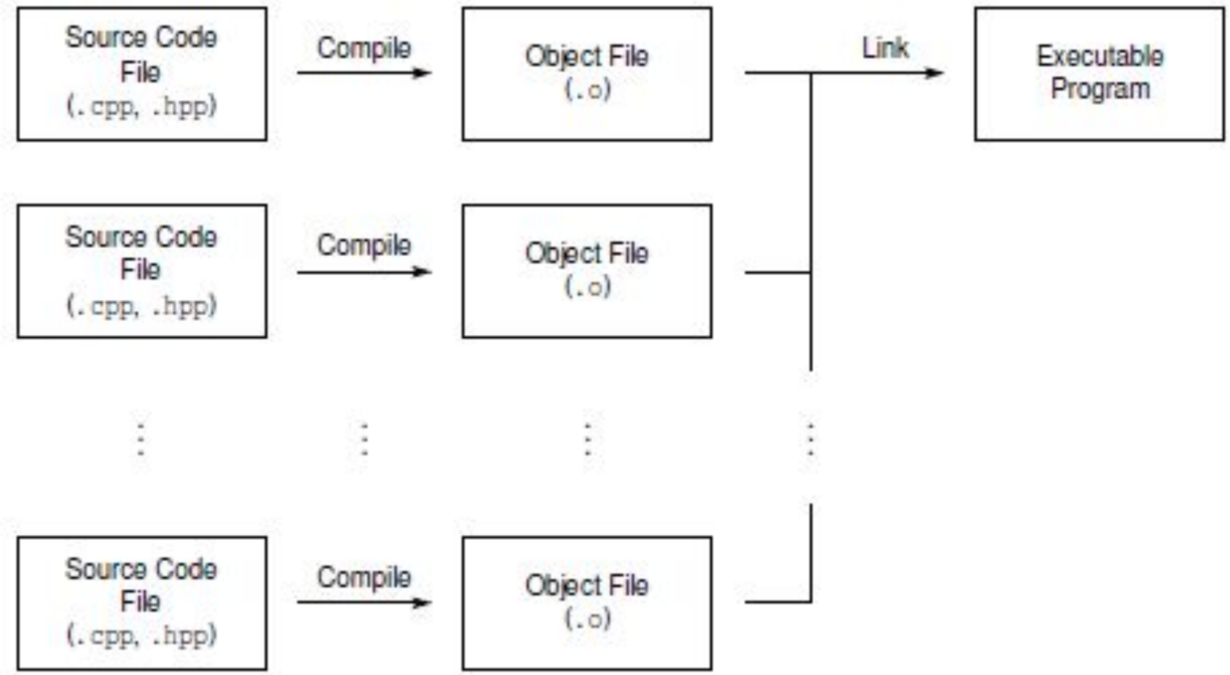
Comments

Header that defines the standard input/output stream objects

starting point for execution

(<<) is called insertion operator

# C++ Build Process Simplified

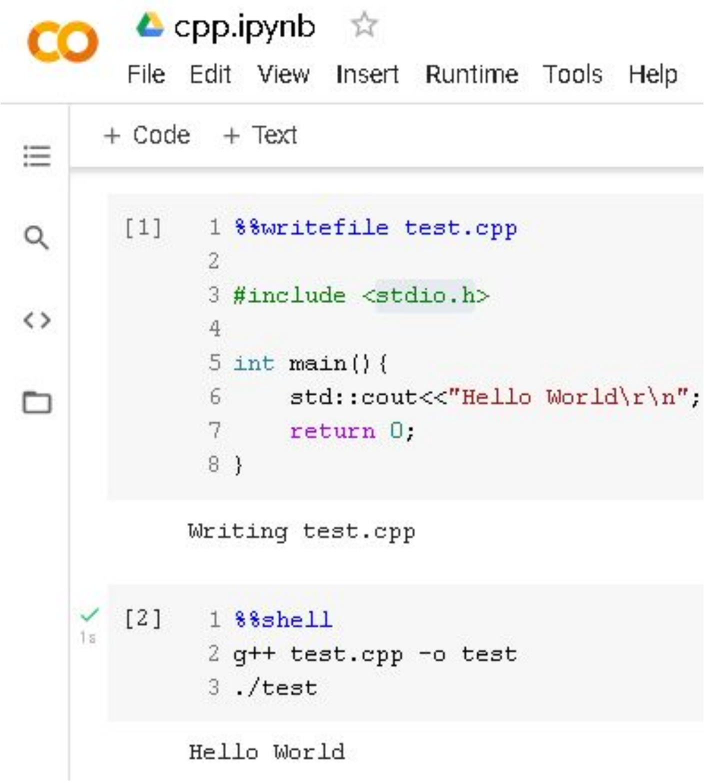


# Running C++ in Google Colab

- ❑ Magic Commands:
  `%%writefile , %%shell , %ls`
- ❑ Compile a C++ source file
  `g++ test.cpp -o outputcpp`
- ❑ Run the execution file
  `./outputcpp`

Create a notebook:

https://colab.research.google.com/#create=true



```
%%writefile test.cpp

#include <stdio.h>

int main(){
    std::cout<<"Hello World\r\n";
    return 0;
}
```

```
Writing test.cpp
```

```
%%shell
g++ test.cpp -o test
./test
```

```
Hello World
```

# Hello World in C++ (printf)

Code

```
#include <stdio.h>

int main(){
    printf("Hello World\r\n");
    return 0;
}
```

Comments

C Header

From C language

# C++ Basics (comments)

- One line comment.
- Multiple lines comment.

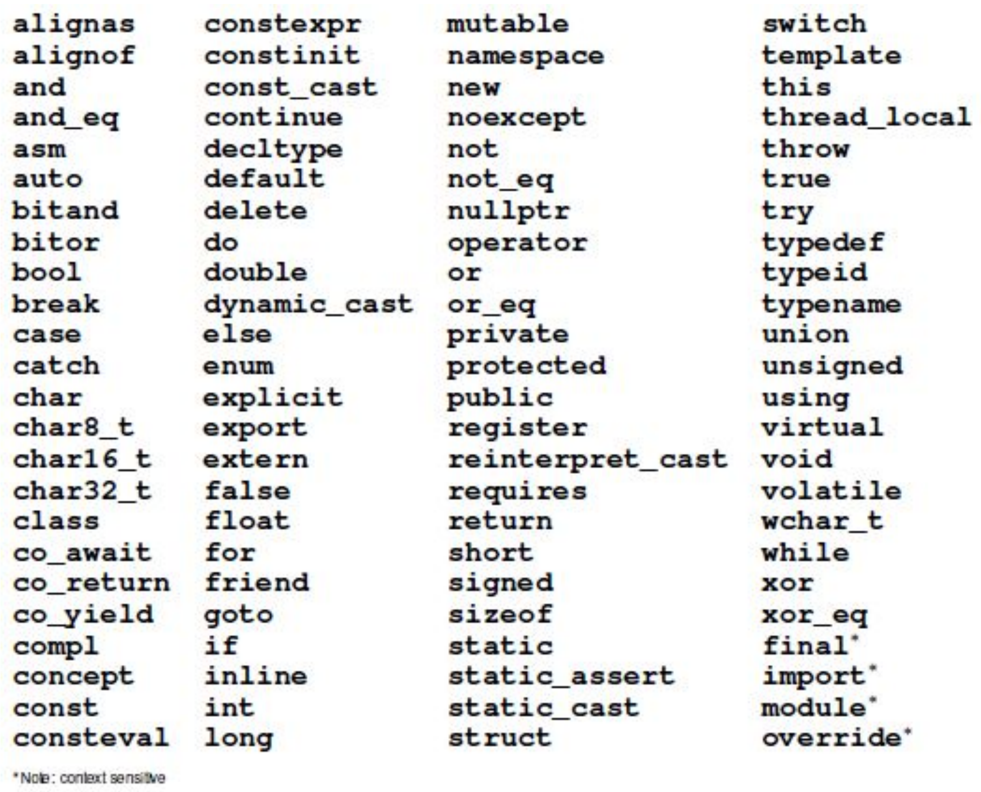
```
//This is one-line comment


/*
This is
multi-line comment
*/
```

# C++ Basics (identifiers)

- ❑ Identifiers are used to name objects, variables, or functions.
- ❑ Valid identifier:
  - ❑ One or more letters and digits.
  - ❑ Can not begin with a digit.
  - ❑ Case sensitive (`var` is not like `Var`).
  - ❑ Not a reserved keyword.
- ❑ Valid Examples:

```
Variable_name

variableName

Variable2

V_A_R_i_b_l_e_1
```

# C++ Basics (reserved keywords)



alignas
alignof
and
and_eq
asm
auto
bitand
bitor
bool
break
case
catch
char
char8_t
char16_t
char32_t
class
co_await
co_return
co_yield
compl
concept
const
consteval
constexpr
constinit
const_cast
continue
decltype
default
delete
do
double
dynamic_cast
else
enum
explicit
export
extern
false
float
for
friend
goto
if
inline
int
long
mutable
namespace
new
noexcept
not
not_eq
nullptr
operator
or
or_eq
private
protected
public
register
reinterpret_cast
requires
return
short
signed
sizeof
static
static_assert
static_cast
struct
switch
template
this
thread_local
throw
true
try
typedef
typeid
typename
union
unsigned
using
virtual
void
volatile
wchar_t
while
xor
xor_eq
final*
import*
module*
override*

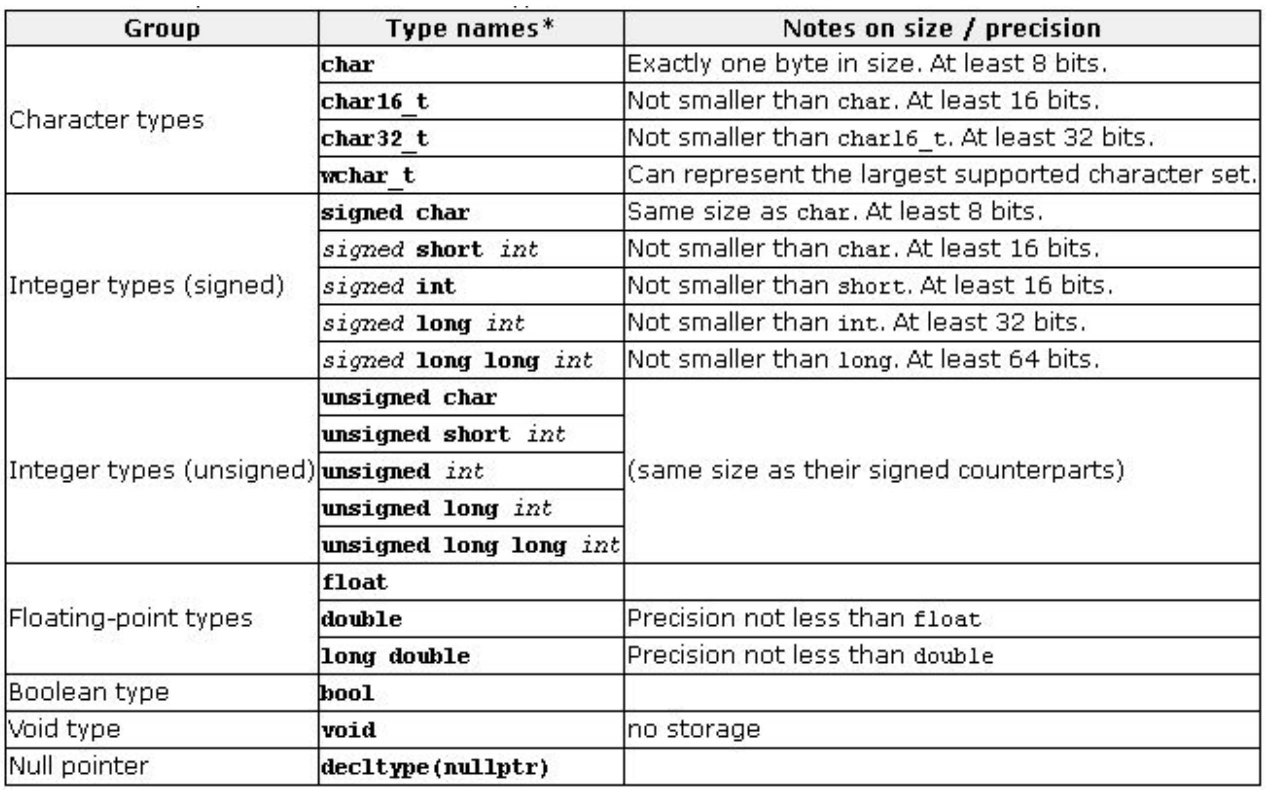
*Note: context sensitive

# C++ Basics (Source-File Inclusion)

- ❑ Preprocessor `#include` directive.
- ❑ Inclusion styles:
  - ❑ `#include <path_specifier>.` Usually for system header files
  - ❑ `#include "path_specifier".` Usually for user-defined header files
- ❑ Another famous preprocessor directive is `#define.` .i.e. `#define pi 3.14`

```cpp
#include <iostream>
#define LANG AR
int main(){
    #if LANG==AR
    std::cout<<"مرحبًا بالعالم\r\n";
    #elif LANG==TR
    std::cout<<"Selam Dünya\r\n";
    #elif LANG==EN
    std::cout<<"Hello World\r\n";
    #endif
    return 0;
}
```

# C++ Basics (variable types)

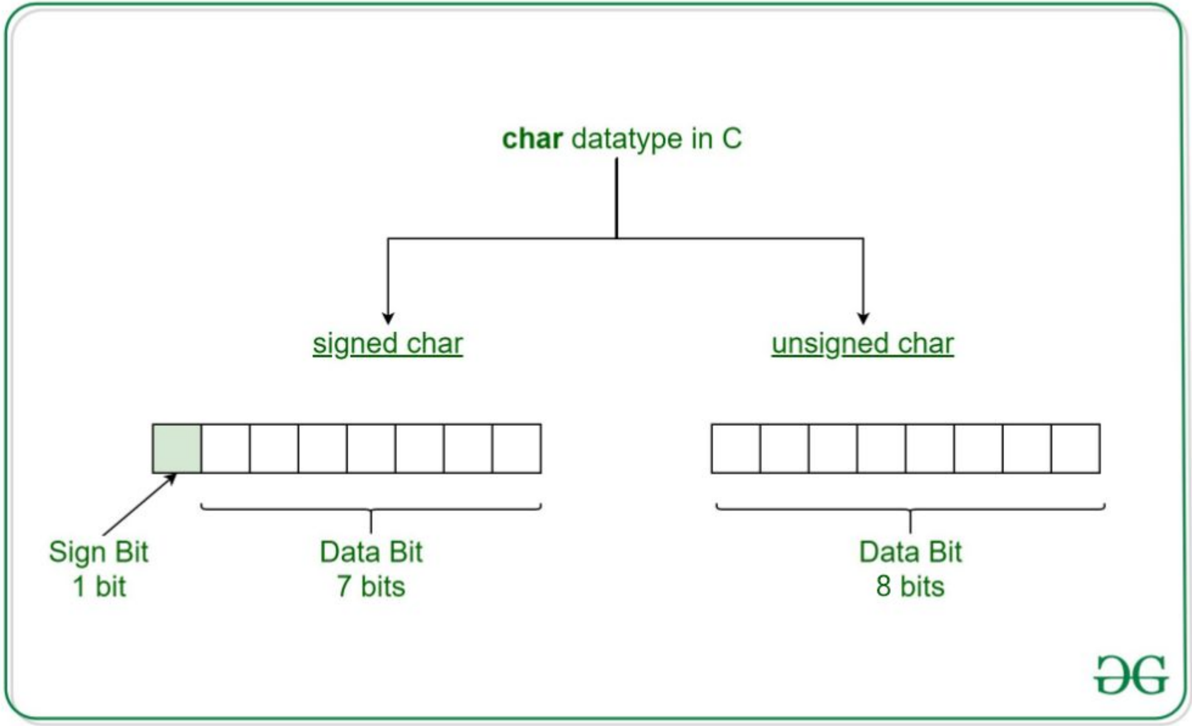


| Group | Type names* | Notes on size / precision |
|---|---|---|
| Character types | char | Exactly one byte in size. At least 8 bits. |
| | char16_t | Not smaller than char. At least 16 bits. |
| | char32_t | Not smaller than char16_t. At least 32 bits. |
| | wchar_t | Can represent the largest supported character set. |
| Integer types (signed) | signed char | Same size as char. At least 8 bits. |
| | *signed* short *int* | Not smaller than char. At least 16 bits. |
| | *signed* int | Not smaller than short. At least 16 bits. |
| | *signed* long *int* | Not smaller than int. At least 32 bits. |
| | *signed* long long *int* | Not smaller than long. At least 64 bits. |
| Integer types (unsigned) | unsigned char | (same size as their signed counterparts) |
| | unsigned short *int* | |
| | unsigned *int* | |
| | unsigned long *int* | |
| | unsigned long long *int* | |
| Floating-point types | float | |
| | double | Precision not less than float |
| | long double | Precision not less than double |
| Boolean type | bool | |
| Void type | void | no storage |
| Null pointer | decltype(nullptr) | |

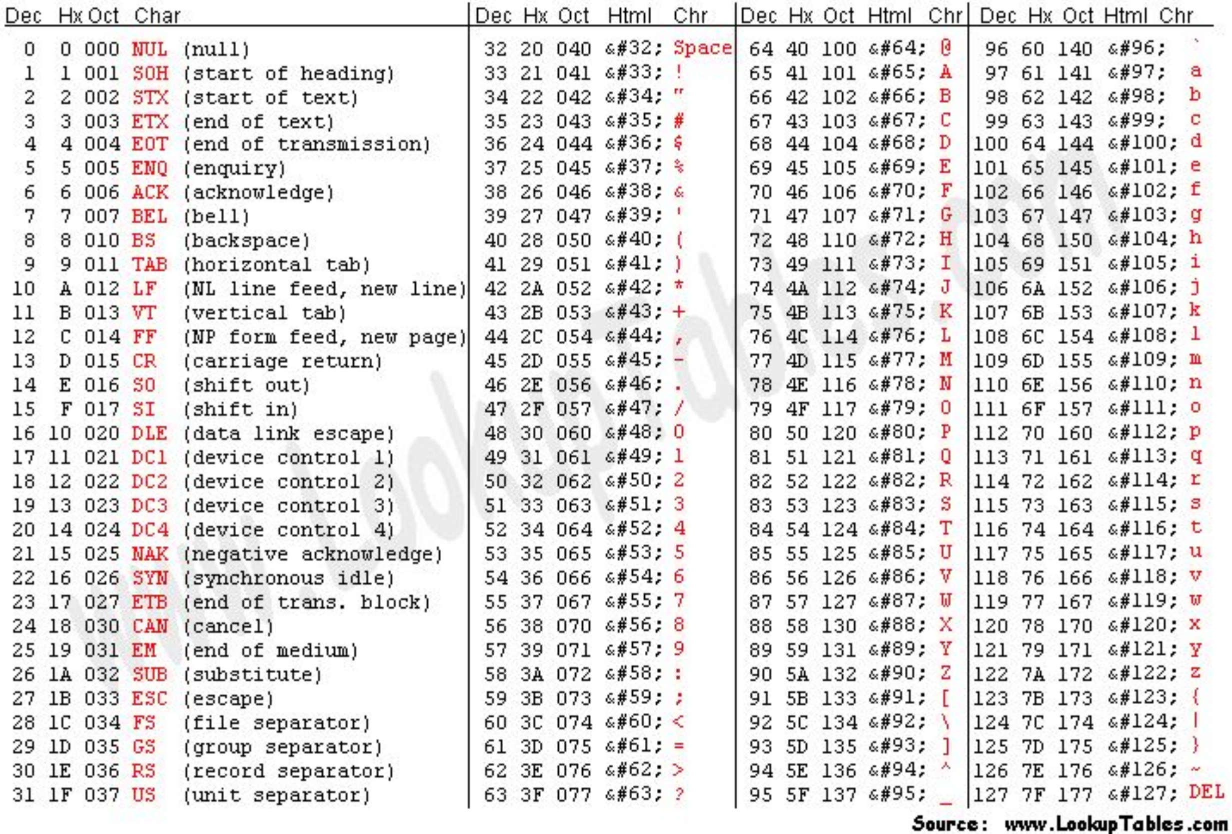
# C++ Basics (variable types)

```cpp
#include <iostream>

int var_int = 16;
char var_char = 'L';
float var_float = 13.4;
bool var_bool = true;
int main(){
    std::cout<<"var_int:"<<var_int<<",size="<<sizeof(var_int)<<std::endl;
    std::cout<<"var_char:"<<var_char<<",size="<<sizeof(var_char)<<std::endl;
    std::cout<<"var_float:"<<var_float<<",size="<<sizeof(var_float)<<std::endl;
    std::cout<<"var_bool:"<<var_bool<<",size="<<sizeof(var_bool)<<std::endl;
    return 0;
}
```

# C++ Basics (variable types: Signed and Unsigned)



# A note about ASCII and char data type



| Dec | Hx | Oct | Char | Dec | Hx | Oct | Html | Chr | Dec | Hx | Oct | Html | Chr | Dec | Hx | Oct | Html | Chr |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 0 | 0 | 000 | NUL (null) | 32 | 20 | 040 | &#32; | Space | 64 | 40 | 100 | &#64; | @ | 96 | 60 | 140 | &#96; | ` |
| 1 | 1 | 001 | SOH (start of heading) | 33 | 21 | 041 | &#33; | ! | 65 | 41 | 101 | &#65; | A | 97 | 61 | 141 | &#97; | a |
| 2 | 2 | 002 | STX (start of text) | 34 | 22 | 042 | &#34; | " | 66 | 42 | 102 | &#66; | B | 98 | 62 | 142 | &#98; | b |
| 3 | 3 | 003 | ETX (end of text) | 35 | 23 | 043 | &#35; | # | 67 | 43 | 103 | &#67; | C | 99 | 63 | 143 | &#99; | c |
| 4 | 4 | 004 | EOT (end of transmission) | 36 | 24 | 044 | &#36; | $ | 68 | 44 | 104 | &#68; | D | 100 | 64 | 144 | &#100; | d |
| 5 | 5 | 005 | ENQ (enquiry) | 37 | 25 | 045 | &#37; | % | 69 | 45 | 105 | &#69; | E | 101 | 65 | 145 | &#101; | e |
| 6 | 6 | 006 | ACK (acknowledge) | 38 | 26 | 046 | &#38; | & | 70 | 46 | 106 | &#70; | F | 102 | 66 | 146 | &#102; | f |
| 7 | 7 | 007 | BEL (bell) | 39 | 27 | 047 | &#39; | ' | 71 | 47 | 107 | &#71; | G | 103 | 67 | 147 | &#103; | g |
| 8 | 8 | 010 | BS (backspace) | 40 | 28 | 050 | &#40; | ( | 72 | 48 | 110 | &#72; | H | 104 | 68 | 150 | &#104; | h |
| 9 | 9 | 011 | TAB (horizontal tab) | 41 | 29 | 051 | &#41; | ) | 73 | 49 | 111 | &#73; | I | 105 | 69 | 151 | &#105; | i |
| 10 | A | 012 | LF (NL line feed, new line) | 42 | 2A | 052 | &#42; | * | 74 | 4A | 112 | &#74; | J | 106 | 6A | 152 | &#106; | j |
| 11 | B | 013 | VT (vertical tab) | 43 | 2B | 053 | &#43; | + | 75 | 4B | 113 | &#75; | K | 107 | 6B | 153 | &#107; | k |
| 12 | C | 014 | FF (NP form feed, new page) | 44 | 2C | 054 | &#44; | , | 76 | 4C | 114 | &#76; | L | 108 | 6C | 154 | &#108; | l |
| 13 | D | 015 | CR (carriage return) | 45 | 2D | 055 | &#45; | - | 77 | 4D | 115 | &#77; | M | 109 | 6D | 155 | &#109; | m |
| 14 | E | 016 | SO (shift out) | 46 | 2E | 056 | &#46; | . | 78 | 4E | 116 | &#78; | N | 110 | 6E | 156 | &#110; | n |
| 15 | F | 017 | SI (shift in) | 47 | 2F | 057 | &#47; | / | 79 | 4F | 117 | &#79; | O | 111 | 6F | 157 | &#111; | o |
| 16 | 10 | 020 | DLE (data link escape) | 48 | 30 | 060 | &#48; | 0 | 80 | 50 | 120 | &#80; | P | 112 | 70 | 160 | &#112; | p |
| 17 | 11 | 021 | DC1 (device control 1) | 49 | 31 | 061 | &#49; | 1 | 81 | 51 | 121 | &#81; | Q | 113 | 71 | 161 | &#113; | q |
| 18 | 12 | 022 | DC2 (device control 2) | 50 | 32 | 062 | &#50; | 2 | 82 | 52 | 122 | &#82; | R | 114 | 72 | 162 | &#114; | r |
| 19 | 13 | 023 | DC3 (device control 3) | 51 | 33 | 063 | &#51; | 3 | 83 | 53 | 123 | &#83; | S | 115 | 73 | 163 | &#115; | s |
| 20 | 14 | 024 | DC4 (device control 4) | 52 | 34 | 064 | &#52; | 4 | 84 | 54 | 124 | &#84; | T | 116 | 74 | 164 | &#116; | t |
| 21 | 15 | 025 | NAK (negative acknowledge) | 53 | 35 | 065 | &#53; | 5 | 85 | 55 | 125 | &#85; | U | 117 | 75 | 165 | &#117; | u |
| 22 | 16 | 026 | SYN (synchronous idle) | 54 | 36 | 066 | &#54; | 6 | 86 | 56 | 126 | &#86; | V | 118 | 76 | 166 | &#118; | v |
| 23 | 17 | 027 | ETB (end of trans. block) | 55 | 37 | 067 | &#55; | 7 | 87 | 57 | 127 | &#87; | W | 119 | 77 | 167 | &#119; | w |
| 24 | 18 | 030 | CAN (cancel) | 56 | 38 | 070 | &#56; | 8 | 88 | 58 | 130 | &#88; | X | 120 | 78 | 170 | &#120; | x |
| 25 | 19 | 031 | EM (end of medium) | 57 | 39 | 071 | &#57; | 9 | 89 | 59 | 131 | &#89; | Y | 121 | 79 | 171 | &#121; | y |
| 26 | 1A | 032 | SUB (substitute) | 58 | 3A | 072 | &#58; | : | 90 | 5A | 132 | &#90; | Z | 122 | 7A | 172 | &#122; | z |
| 27 | 1B | 033 | ESC (escape) | 59 | 3B | 073 | &#59; | ; | 91 | 5B | 133 | &#91; | [ | 123 | 7B | 173 | &#123; | { |
| 28 | 1C | 034 | FS (file separator) | 60 | 3C | 074 | &#60; | < | 92 | 5C | 134 | &#92; | \ | 124 | 7C | 174 | &#124; | \| |
| 29 | 1D | 035 | GS (group separator) | 61 | 3D | 075 | &#61; | = | 93 | 5D | 135 | &#93; | ] | 125 | 7D | 175 | &#125; | } |
| 30 | 1E | 036 | RS (record separator) | 62 | 3E | 076 | &#62; | > | 94 | 5E | 136 | &#94; | ^ | 126 | 7E | 176 | &#126; | ~ |
| 31 | 1F | 037 | US (unit separator) | 63 | 3F | 077 | &#63; | ? | 95 | 5F | 137 | &#95; | _ | 127 | 7F | 177 | &#127; | DEL |

Source: www.LookupTables.com

# A note about ASCII and char data type

```
#include <iostream>

char var_char = '9';

int main(){

    int num = var_char - '0'
    cout<<num;
    return 0;
}
```

# C++ Basics (Declarations and Definitions)

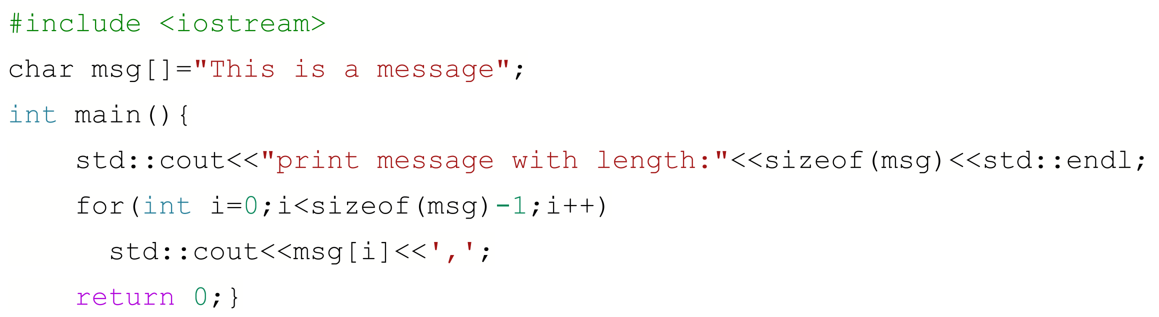
- ❑ Example: Function declaration specifies number of parameters, type of each parameter, and type of return value.
- ❑ Definition provides full information about identifier.
- ❑ Can declare identifier multiple times but can define only once

```
bool isOdd(int x); // declare isOdd

bool isOdd(int x) { // define isOdd
return x % 2;
}
```

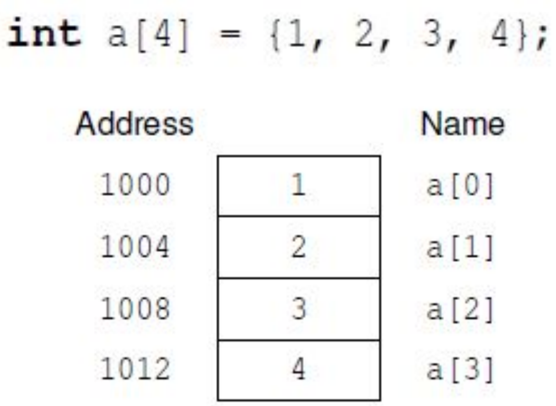
```
struct Vector2 { // declare & define Vector2
double x;
double y;
};
```

# C++ Basics (Arrays)

- ❑ Array is a sequence of one or more item of same type.
- ❑ Array variable is denoted by []
- ❑ Compiler can set the array size at compile time if not provided
- ❑ Example:
    - ❑ `Int x[10]; // array of 10 integers`
    - ❑ `int data[512][512]; // 512 by 512 array of ints`
    - ❑ `char msg[]=”This is a message”`



```
#include <iostream>
char msg[]="This is a message";
int main(){
    std::cout<<"print message with length:"<<sizeof(msg)<<std::endl;
    for(int i=0;i<sizeof(msg)-1;i++)
      std::cout<<msg[i]<<',';
    return 0;}
```



`int a[4] = {1, 2, 3, 4};`

| Address |  | Name |
|---|---|---|
| 1000 | 1 | a[0] |
| 1004 | 2 | a[1] |
| 1008 | 3 | a[2] |
| 1012 | 4 | a[3] |

# C++ Basics (Arrays): 2D matrices multiplication

```
#include <iostream>
char arr1[3][3]={{2,3,5},{4,5,6},{10,1,0}};
char arr2[3][3]={{1,0,0},{1,1,0},{1,1,1}};
char arr3[3][3]={0};
int main(){
    for(int i=0;i<3;i++)
    {
      for(int j=0;j<3;j++)
      {
        for(int k=0;k<3;k++)
        {
         arr3[i][j] += (arr1[i][k] * arr2[k][j]) ;
        }
       std::cout<<int(arr3[i][j])<<" ";
      }
      std::cout<<std::endl;
    }return 0;}
```

# C++ Basics (Pointers)

- Pointer is data type that stores only addresses in memory.
- Pointer to data type T is denoted by T*. .i.e `int * ptr`
- ***Dereferencing*** is accessing object to which pointer refers.
- The operator to access the address of an item is `&`. .i.e

  `int * ptr = &var;`

```
#include <iostream>
    int var = 5;
    int* ptr = &var;

int main(){

    *ptr = 10 ;
    std::cout<<var;
    return 0;}
```

# C++ Basics (Pointers)

- Pointer is data type that stores only addresses in memory.
- Pointer to data type T is denoted by T*. .i.e `int * ptr`
- ***Dereferencing*** is accessing object to which pointer refers.
- The operator to access the address of an item is `&`. .i.e

```
int * ptr = &var;
```

```
#include <iostream>
    int var = 5;
    int* ptr = &var;

int main(){

    *ptr = 10 ;
    std::cout<<var;
    return 0;}
```

# C++ Basics (Pointers)

```cpp
#include <iostream>

    int var = 0xAABBCCDD;
    char* ptr = (char*)&var;

int main(){

    ptr[0] = 0;

    std::cout<<std::hex<<var;

    return 0;}
```

# C++ Basics (typedef)

- User-defined data type can be defined using `typedef` keyword.
- Example:

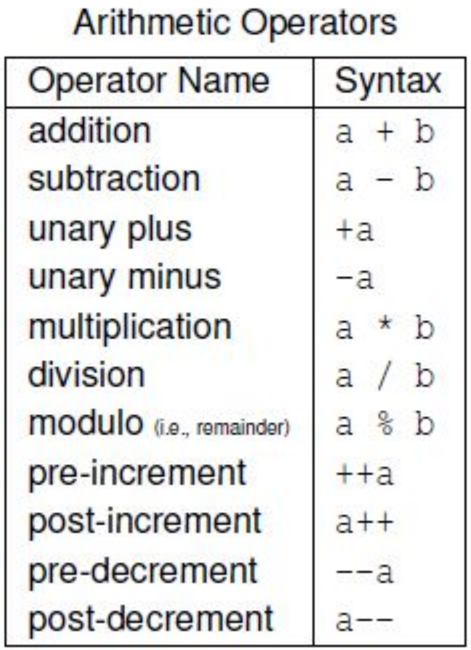
```
typedef char* CharPtr;

CharPtr p; // p has type char*
```

# C++ Basics (auto)

- `auto` can be used if you want the compiler to decide the variable type based on initialization value.
- Example:

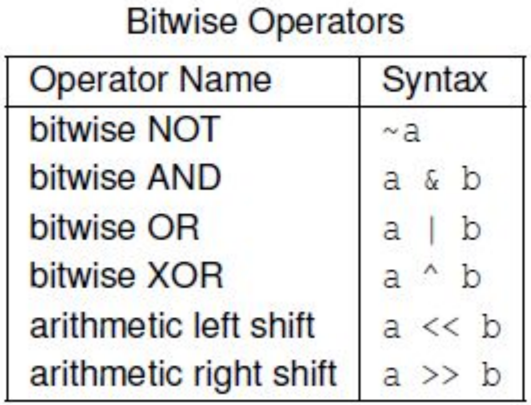
```
auto i = 3; // i has type int
auto j = i; // j has type int
auto& k = i; // k has type int&
auto x = 3.14; // x has type double
```

# C++ Basics (operators)



Arithmetic Operators

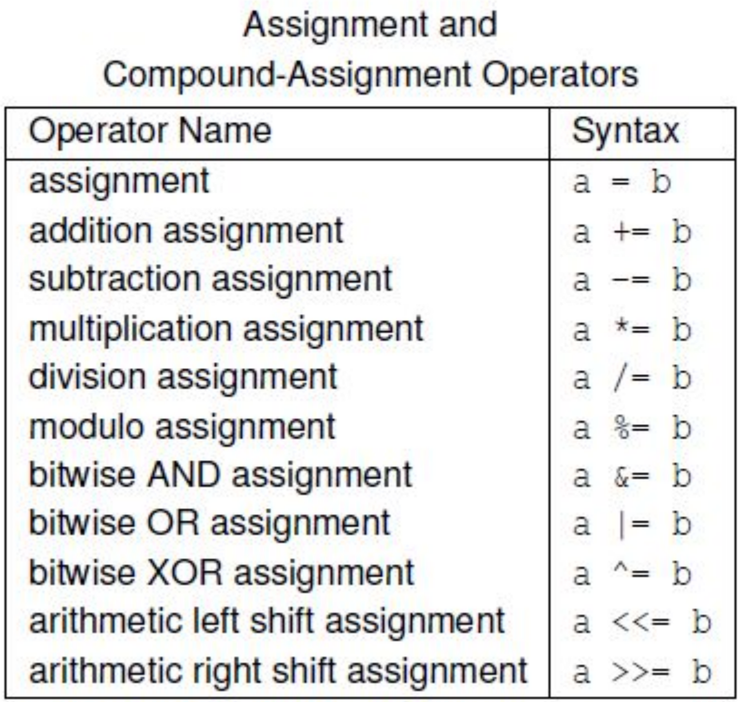
| Operator Name | Syntax |
|---|---|
| addition | `a + b` |
| subtraction | `a - b` |
| unary plus | `+a` |
| unary minus | `-a` |
| multiplication | `a * b` |
| division | `a / b` |
| modulo (i.e., remainder) | `a % b` |
| pre-increment | `++a` |
| post-increment | `a++` |
| pre-decrement | `--a` |
| post-decrement | `a--` |



Bitwise Operators

| Operator Name | Syntax |
|---|---|
| bitwise NOT | `~a` |
| bitwise AND | `a & b` |
| bitwise OR | `a \| b` |
| bitwise XOR | `a ^ b` |
| arithmetic left shift | `a << b` |
| arithmetic right shift | `a >> b` |

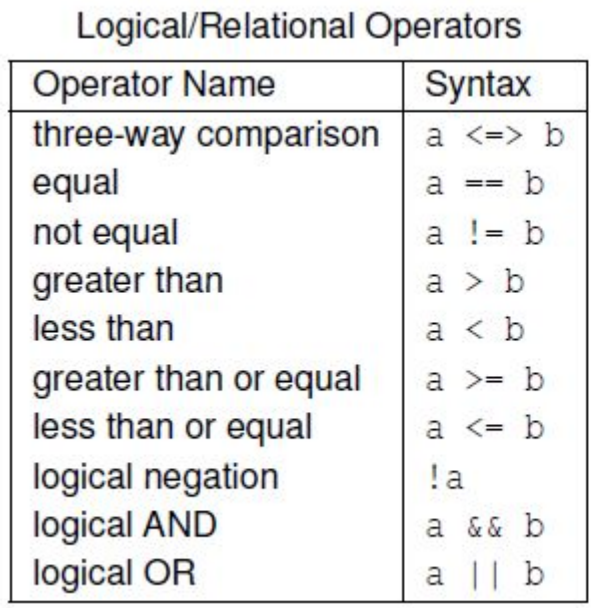
# C++ Basics (operators)



Assignment and
Compound-Assignment Operators

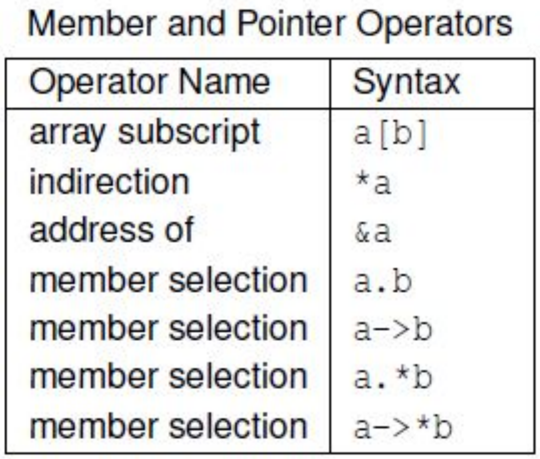
| Operator Name | Syntax |
|---|---|
| assignment | `a = b` |
| addition assignment | `a += b` |
| subtraction assignment | `a -= b` |
| multiplication assignment | `a *= b` |
| division assignment | `a /= b` |
| modulo assignment | `a %= b` |
| bitwise AND assignment | `a &= b` |
| bitwise OR assignment | `a \|= b` |
| bitwise XOR assignment | `a ^= b` |
| arithmetic left shift assignment | `a <<= b` |
| arithmetic right shift assignment | `a >>= b` |

# C++ Basics (operators)



Logical/Relational Operators

| Operator Name | Syntax |
|---|---|
| three-way comparison | `a <=> b` |
| equal | `a == b` |
| not equal | `a != b` |
| greater than | `a > b` |
| less than | `a < b` |
| greater than or equal | `a >= b` |
| less than or equal | `a <= b` |
| logical negation | `!a` |
| logical AND | `a && b` |
| logical OR | `a \|\| b` |



Member and Pointer Operators

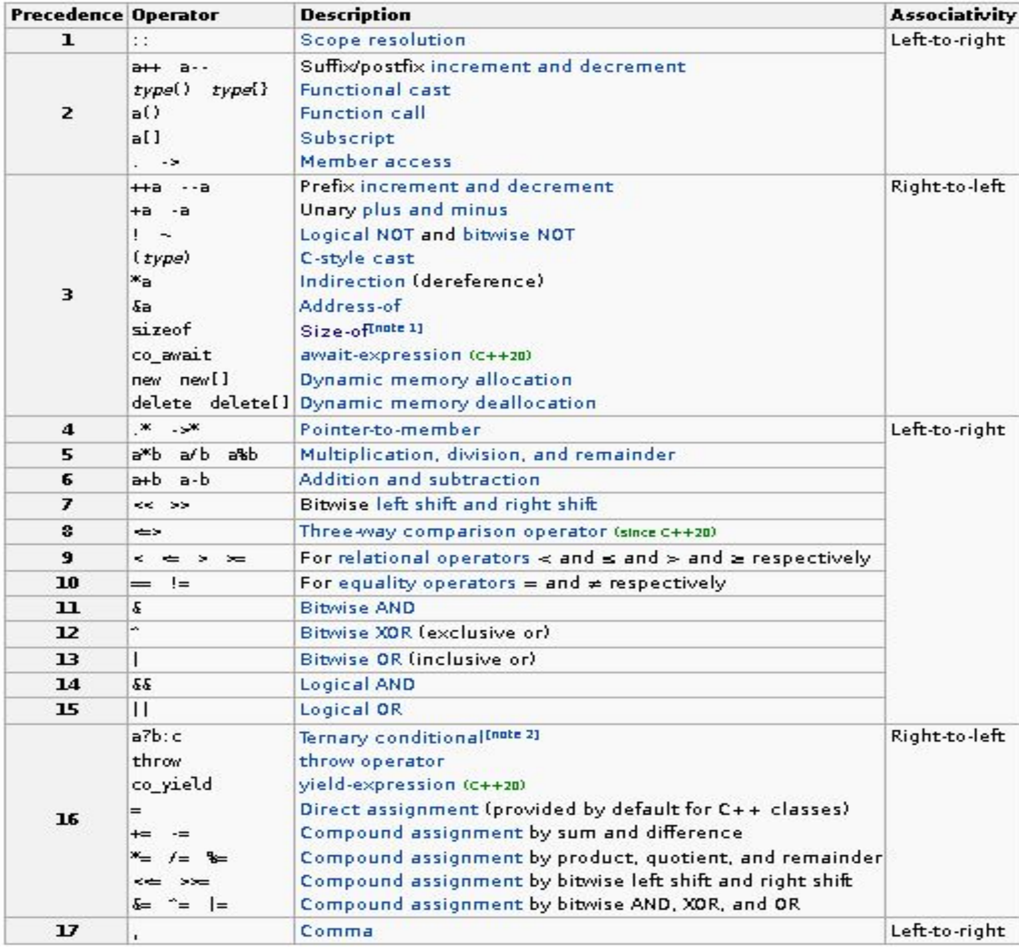
| Operator Name | Syntax |
|---|---|
| array subscript | `a[b]` |
| indirection | `*a` |
| address of | `&a` |
| member selection | `a.b` |
| member selection | `a->b` |
| member selection | `a.*b` |
| member selection | `a->*b` |

# C++ Basics (Operator Precedence)

Examples:

```
a || b && c || d
```

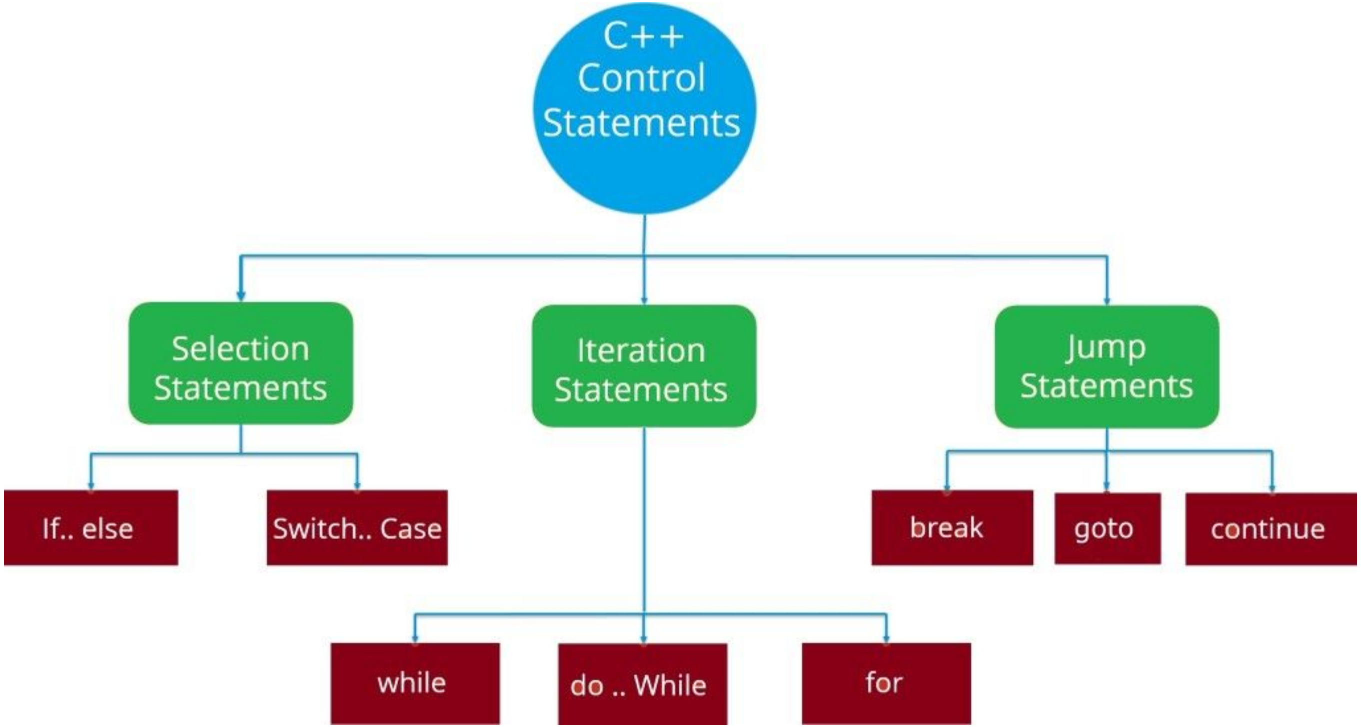
```
z *= ++y + 5
```



| Precedence | Operator | Description | Associativity |
|---|---|---|---|
| 1 | :: | Scope resolution | Left-to-right |
| 2 | a++ a--<br>*type*() *type*{}<br>a()<br>a[]<br>. -> | Suffix/postfix increment and decrement<br>Functional cast<br>Function call<br>Subscript<br>Member access | |
| 3 | ++a --a<br>+a -a<br>! ~<br>(*type*)<br>*a<br>&a<br>sizeof<br>co_await<br>new new[]<br>delete delete[] | Prefix increment and decrement<br>Unary plus and minus<br>Logical NOT and bitwise NOT<br>C-style cast<br>Indirection (dereference)<br>Address-of<br>Size-of[note 1]<br>await-expression (C++20)<br>Dynamic memory allocation<br>Dynamic memory deallocation | Right-to-left |
| 4 | .* ->* | Pointer-to-member | Left-to-right |
| 5 | a*b a/b a%b | Multiplication, division, and remainder | |
| 6 | a+b a-b | Addition and subtraction | |
| 7 | << >> | Bitwise left shift and right shift | |
| 8 | <=> | Three-way comparison operator (since C++20) | |
| 9 | < <= > >= | For relational operators < and ≤ and > and ≥ respectively | |
| 10 | == != | For equality operators = and ≠ respectively | |
| 11 | & | Bitwise AND | |
| 12 | ^ | Bitwise XOR (exclusive or) | |
| 13 | \| | Bitwise OR (inclusive or) | |
| 14 | && | Logical AND | |
| 15 | \|\| | Logical OR | |
| 16 | a?b:c<br>throw<br>co_yield<br>=<br>+= -=<br>*= /= %=<br><<= >>=<br>&= ^= \|= | Ternary conditional[note 2]<br>throw operator<br>yield-expression (C++20)<br>Direct assignment (provided by default for C++ classes)<br>Compound assignment by sum and difference<br>Compound assignment by product, quotient, and remainder<br>Compound assignment by bitwise left shift and right shift<br>Compound assignment by bitwise AND, XOR, and OR | Right-to-left |
| 17 | , | Comma | Left-to-right |

https://en.cppreference.com/w/cpp/language/operator_precedence

# C++ Basics (**Control Flow**)



https://www.w3adda.com/cplusplus-tutorial/cpp-control-flow-statements

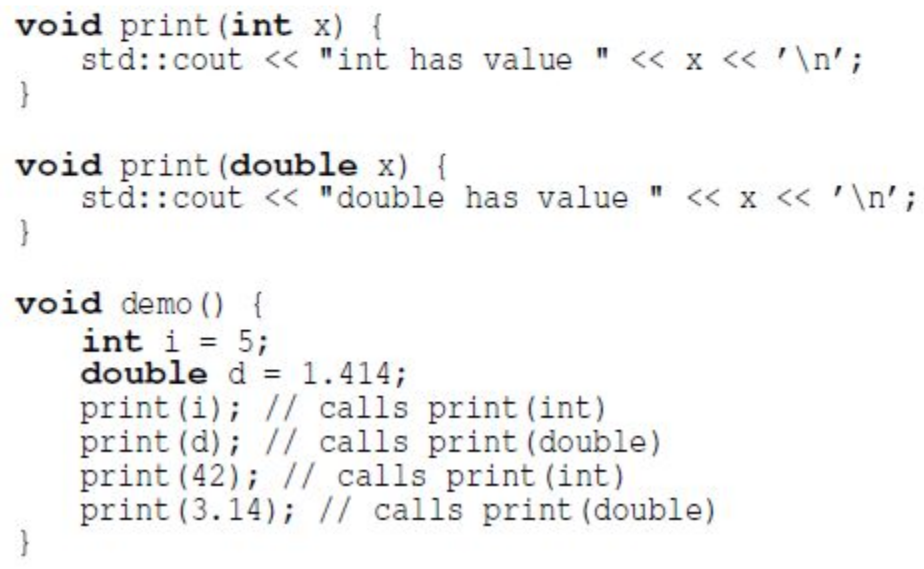
# C++ Basics (main function arguments)

```
#include <iostream>

int main(int argc, char*
argv[]){
    std::cout<<"number of
argument:"<<argc<<"\r\narg
1:"<<argv[1]<<"\r\narg2:"<
<argv[2];
    return 0;
}
```

```
g++ test.cpp -o test
./test arg1 arg2
```

# C++ Basics (**function overloading**)

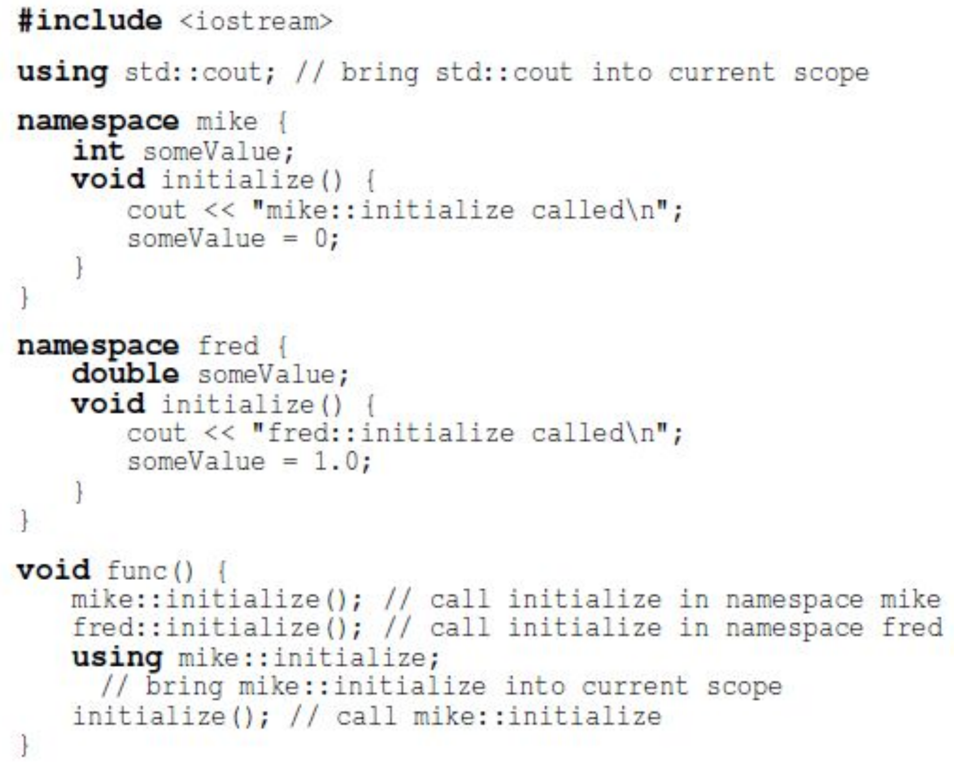
- Function overloading is having the same function name but with different definition based on arguments (type and count).



```cpp
void print(int x) {
    std::cout << "int has value " << x << '\n';
}

void print(double x) {
    std::cout << "double has value " << x << '\n';
}

void demo() {
    int i = 5;
    double d = 1.414;
    print(i); // calls print(int)
    print(d); // calls print(double)
    print(42); // calls print(int)
    print(3.14); // calls print(double)
}
```

# C++ Basics (**Namespaces**)

- ❑ `namespace` is a region that provides scope for identifiers declared inside.
- ❑ scope-resolution operator (::) specify namespace to which particular identifier belongs.



```
#include <iostream>

using std::cout; // bring std::cout into current scope

namespace mike {
    int someValue;
    void initialize() {
        cout << "mike::initialize called\n";
        someValue = 0;
    }
}

namespace fred {
    double someValue;
    void initialize() {
        cout << "fred::initialize called\n";
        someValue = 1.0;
    }
}

void func() {
    mike::initialize(); // call initialize in namespace mike
    fred::initialize(); // call initialize in namespace fred
    using mike::initialize;
      // bring mike::initialize into current scope
    initialize(); // call mike::initialize
}
```

# Assignment 2: Exercise 1

- Convert letters from A to Z from capital letter to small letter. Start using the following code.

```
#include <iostream>

char captital_letters[]={/* your code here*/};

int main(){

    for(int i=0;/*your code here*/;i++)
      // your code here "cout<<...";

    return 0;
}
```

# Assignment 2: Exercise 2

- Write a program to do 2D matrix multiplication. $Mat1_{r1\times c1}, Mat2_{r2\times c2}$. User should enter first the dimensions (through an error using `assert` if C1 != r2). Then user enters the values of each matrix.

# Assignment 2: Exercise 3

- Let be `var=0xDEAD0000` , using a pointer of type char* pointing to var. Change the value to `0xDEADBEAF`.

# Assignment 2: Exercise 1 Solution

```
#include <iostream>

char
captital_letters[]={'A','B','C','D','E','F','G','H','I','J','K','L','M','N','O',
'P','Q','R','S','T','U','V','W','X','Y','Z'};

int main(){

    for(int i=0;i<sizeof(captital_letters);i++)
      std::cout<<captital_letters[i]-' '<<',';

    return 0;
}
```