Variables and Operators

Ahsan Ayub

Ph.D. Student, Department of Computer Science

Graduate Research Assistant, CEROC

CSC 1300: Introduction to Programming

Tuesday, August 24, 2021

Variables

- A container (storage area) to hold data
- Can only hold one thing at a time
- The contents of the container (variable) may change or vary
- Must be defined with a statement (called a variable definition)

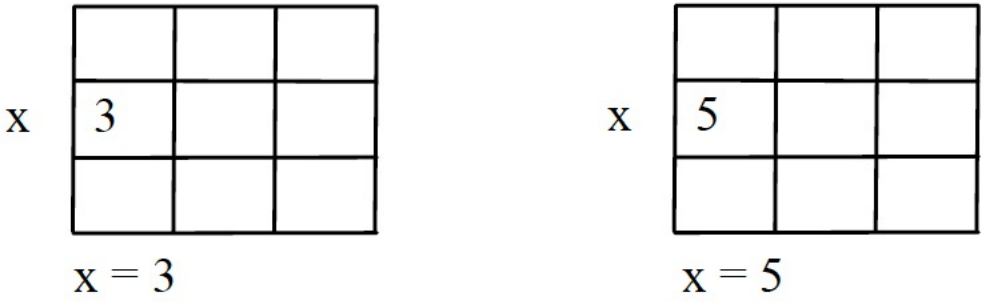


Image Source: “Let Us C” by Yashwant Kanetkar

Declaring a Variable

```
int LaptopPrice;
```

```
double LaptopPriceWithTax, SalesTax;
```

Initializing a Variable

```
int LaptopPrice; // Declaration
LaptopPrice = 1099; // Initialization
```

```
// Declaration and Initialization at the
// same statement
int LaptopPrice = 1099;
```

```
// Declaration
double LaptopPriceWithTax, SalesTax;
// Initialization
LaptopPriceWithTax = 1208.90;
SalesTax = 0.0975;
```

```
// Declaration and Initialization at the same statement
double LaptopPriceWithTax = 1208.90, SalesTax = 0.0975;
```

Floating-point Literals

```
// Declaration of variables
double LaptopPriceWithTax, SalesTax;
// Decimal notations to initialize
LaptopPriceWithTax = 1208.90;
SalesTax = 0.0975;
```

```
// Declaration of variables
double LaptopPriceWithTax, SalesTax;
// E notation to initialize
LaptopPriceWithTax = 1.2089E3;
SalesTax = 9.75e-5;
```

Constant Variables

- Constant variables’ values are fixed for the duration of the program.
- We use `const` keyword to define the variable.

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

int main()
{
        // Declaration and initialization of the constant variable
        // Variable’s name could also be declared as “SALES_TAX”
        const double kSalesTax = 0.0975;
        // Declaration and initialization of the other variables
        int LaptopPrice = 1099;
        double LaptopPriceWithTax = 1208.90;
        kSalesTax = 0.0675; // Error – cannot be modified
        return 0;
}
```

Further Reading on Naming Convention: https://google.github.io/styleguide/cppguide.html#Constant_Names

C++ Operators

- Arithmetic Operators
- Assignment Operators
- Relational Operators
- Logical Operators
- Bitwise Operators
- Other Operators

C++ Arithmetic Operators

- Addition
- Subtraction
- Multiplication
- Division
- Modulo (remainder after division)

Output Tracing – Addition

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

int main()
{
        // Declaration and initialization of the variables
        int x = 10, y = 5, sum;
        sum = x + y;    // Performing the addition
        cout << sum << endl;
        return 0;
}
```

```
??
```

Output Tracing – Addition

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

int main()
{
        // Declaration and initialization of the variables
        int x = 10, y = 5, sum;
        sum = x + y;    // Performing the addition
        cout << sum << endl;
        return 0;
}
```

```
15
```

Output Tracing – Subtraction

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

int main()
{
        // Declaration and initialization of the variables
        int x = 10, y = 15, sub;
        sub = x - y;    // Performing the subtraction
        cout << sub << endl;
        return 0;
}
```

```
??
```

Output Tracing – Subtraction

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

int main()
{
        // Declaration and initialization of the variables
        int x = 10, y = 15, sub;
        sub = x - y;    // Performing the subtraction
        cout << sub << endl;
        return 0;
}
```

```
-5
```

Output Tracing – Subtraction

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

int main()
{
        // Declaration and initialization of the variables
        int x = 10, y = 15;
        unsigned int sub;
        sub = x - y;     // Performing the subtraction
        cout << sub << endl;
        return 0;
}
```

```
??
```

Output Tracing – Subtraction

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

int main()
{
        // Declaration and initialization of the variables
        int x = 10, y = 15;
        unsigned int sub;
        sub = x - y;    // Performing the subtraction
        cout << sub << endl;
        return 0;
}
```

```
4,294,967,291
```

Range for the *unsigned* integers is 0 to $2^{32} - 1$

Output Tracing – Multiplication

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

int main()
{
        // Declaration and initialization of the variables
        int x = 10, y = 5, mul;
        mul = x * y;     // Performing the multiplication
        cout << mul << endl;
        return 0;
}
```

```
??
```

Output Tracing – Multiplication

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

int main()
{
        // Declaration and initialization of the variables
        int x = 10, y = 5, mul;
        mul = x * y;     // Performing the multiplication
        cout << mul << endl;
        return 0;
}
```

```
50
```

Output Tracing – Division

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

int main()
{
        // Declaration and initialization of the variables
        double x = 6, y = 4, div;
        div = x / y;     // Performing the division
        cout << div << endl;
        return 0;
}
```

```
??
```

Output Tracing – Division

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

int main()
{
        // Declaration and initialization of the variables
        double x = 6, y = 4, div;
        div = x / y;     // Performing the division
        cout << div << endl;
        return 0;
}
```

```
1.5
```

Output Tracing – Division

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

int main()
{
        // Declaration and initialization of the variables
        int x = 6, y = 4;
        double div;
        div = x / y;     // Performing the division
        cout << div << endl;
        return 0;
}
```

```
??
```

Output Tracing – Division

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

int main()
{
        // Declaration and initialization of the variables
        int x = 6, y = 4;
        double div;
        div = x / y;     // Performing the division
        cout << div << endl;
        return 0;
}
```

```
1
```

Output Tracing – Division

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

int main()
{
        // Declaration and initialization of the variables
        int x = 6;
        double y = 4, div;
        div = x / y;     // Performing the division
        cout << div << endl;
        return 0;
}
```

```
??
```

Output Tracing – Division

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

int main()
{
        // Declaration and initialization of the variables
        int x = 6;
        double y = 4, div;
        div = x / y;     // Performing the division
        cout << div << endl;
        return 0;
}
```

```
1.5
```

Output Tracing – Division

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

int main()
{
        // Declaration and initialization of the variables
        int x = 6, y = 4;
        double div = x / y;      // Performing the division
        cout << div << endl;
        return 0;
}
```

```
??
```

Output Tracing – Division

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

int main()
{
        // Declaration and initialization of the variables
        int x = 6, y = 4;
        double div = x / y;      // Performing the division
        cout << div << endl;
        return 0;
}
```

```
1
```

Output Tracing – Division

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

int main()
{
	// Demonstration of multiple combinations of division
	cout << 6 / 4 << endl;
	cout << 6.0 / 4 << endl;
	cout << 6 / 4.0 << endl;
	cout << 6.0 / 4.0 << endl;
	return 0;
}
```

```
1
1.5
1.5
1.5
```

Output Tracing – Modulus

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

int main()
{
        // Declaration and initialization of the variables
        int number = 113, mod;
        mod = number % 2;        // Performing the modulus operation
        cout << mod << endl;
        return 0;
}
```

```
??
```

Output Tracing – Modulus

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

int main()
{
        // Declaration and initialization of the variables
        int number = 113, mod;
        mod = number % 2;        // Performing the modulus operation
        cout << mod << endl;
        return 0;
}
```

```
1
```

Output Tracing – Modulus

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

int main()
{
        // Declaration and initialization of the variables
        int number = 226, mod;
        mod = number % 2;        // Performing the modulus operation
        cout << mod << endl;
        return 0;
}
```

```
??
```

Output Tracing – Modulus

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

int main()
{
        // Declaration and initialization of the variables
        int number = 226, mod;
        mod = number % 2;       // Performing the modulus operation
        cout << mod << endl;
        return 0;
}
```

```
0
```

Increment and Decrement

- Prefix increment and decrement
  - Pre-increment (++expr)
  - Pre-decrement (--expr)
- Variable’s value is first incremented and then used inside the expression.
- Postfix increment and decrement
  - Post-increment (expr++)
  - Pre-increment (expr--)
- Variable’s value is first used in a expression and then incremented

```
int x = 10, a;
// x = x + 1 and then a = x (modified)
a = ++x;
cout << “x: ” << x << “, a: ” << a << endl;
```

```
x: 11, a: 11
```

```
int x = 10, a;
// a = x (old) and then x = x + 1
a = x++;
cout << “x: ” << x << “, a: ” << a << endl;
```

```
x: 11, a: 10
```

Further Reading:

(1) https://en.cppreference.com/w/cpp/language/operator_incdec

(2) https://www.geeksforgeeks.org/pre-increment-and-post-increment-in-c/

Sizeof Operator

- `sizeof` operator queries the size of the object or type

```
sizeof(int); // Returns 4 (hint: 4 bytes)
```

```
double myVar = 10.6;
sizeof(myVar); // Returns 8 (hint: 8 bytes)
```

Further Reading (Optional): https://en.cppreference.com/w/cpp/language/sizeof

C++ Assignment Operators

Operator	Example	Equivalent to	Value of res for x = 5
`=`	`res = x`	`res = x`	
`+=`	`res += x`	`res = res + x`	
`-=`	`res -= x`	`res = res - x`	
`*=`	`res *= x`	`res = res * x`	
`/=`	`res /= x`	`res = res / x`	
`%=`	`res %= x`	`res = res % x`	

C++ Assignment Operators

Operator	Example	Equivalent to	Value of res for x = 5
`=`	`res = x`	`res = x`	5
`+=`	`res += x`	`res = res + x`	
`-=`	`res -= x`	`res = res - x`	
`*=`	`res *= x`	`res = res * x`	
`/=`	`res /= x`	`res = res / x`	
`%=`	`res %= x`	`res = res % x`	

C++ Assignment Operators

Operator	Example	Equivalent to	Value of res for x = 5
`=`	`res = x`	`res = x`	5
`+=`	`res += x`	`res = res + x`	10
`-=`	`res -= x`	`res = res - x`	
`*=`	`res *= x`	`res = res * x`	
`/=`	`res /= x`	`res = res / x`	
`%=`	`res %= x`	`res = res % x`	

C++ Assignment Operators

Operator	Example	Equivalent to	Value of res for x = 5
`=`	`res = x`	`res = x`	5
`+=`	`res += x`	`res = res + x`	10
`-=`	`res -= x`	`res = res - x`	5
`*=`	`res *= x`	`res = res * x`	
`/=`	`res /= x`	`res = res / x`	
`%=`	`res %= x`	`res = res % x`	

C++ Assignment Operators

Operator	Example	Equivalent to	Value of res for x = 5
`=`	`res = x`	`res = x`	5
`+=`	`res += x`	`res = res + x`	10
`-=`	`res -= x`	`res = res - x`	5
`*=`	`res *= x`	`res = res * x`	25
`/=`	`res /= x`	`res = res / x`	
`%=`	`res %= x`	`res = res % x`	

C++ Assignment Operators

Operator	Example	Equivalent to	Value of res for x = 5
`=`	`res = x`	`res = x`	5
`+=`	`res += x`	`res = res + x`	10
`-=`	`res -= x`	`res = res - x`	5
`*=`	`res *= x`	`res = res * x`	25
`/=`	`res /= x`	`res = res / x`	5
`%=`	`res %= x`	`res = res % x`	

C++ Assignment Operators

Operator	Example	Equivalent to	Value of res for x = 5
`=`	`res = x`	`res = x`	5
`+=`	`res += x`	`res = res + x`	10
`-=`	`res -= x`	`res = res - x`	5
`*=`	`res *= x`	`res = res * x`	25
`/=`	`res /= x`	`res = res / x`	5
`%=`	`res %= x`	`res = res % x`	0

C++ Relational Operators

Operator	Meaning	Example	Value of res for a = 3 and b = 5
`==`	Is equal to	`bool res = a == b`	
`!=`	Not equal to	`bool res = a != b`	
`>`	Greater than	`bool res = a > b`	
`<`	Less than	`bool res = a < b`	
`>=`	Greater than or equal to	`bool res = a >= b`	
`<=`	Less than or equal to	`bool res = a <= b`	

C++ Relational Operators

Operator	Meaning	Example	Value of res for a = 3 and b = 5
`==`	Is equal to	`bool res = a == b`	`false`
`!=`	Not equal to	`bool res = a != b`	
`>`	Greater than	`bool res = a > b`	
`<`	Less than	`bool res = a < b`	
`>=`	Greater than or equal to	`bool res = a >= b`	
`<=`	Less than or equal to	`bool res = a <= b`	

C++ Relational Operators

Operator	Meaning	Example	Value of res for a = 3 and b = 5
==	Is equal to	bool res = a == b	false
!=	Not equal to	bool res = a != b	true
>	Greater than	bool res = a > b	
<	Less than	bool res = a < b	
>=	Greater than or equal to	bool res = a >= b	
<=	Less than or equal to	bool res = a <= b	

C++ Relational Operators

Operator	Meaning	Example	Value of res for a = 3 and b = 5
`==`	Is equal to	`bool res = a == b`	`false`
`!=`	Not equal to	`bool res = a != b`	`true`
`>`	Greater than	`bool res = a > b`	`false`
`<`	Less than	`bool res = a < b`	
`>=`	Greater than or equal to	`bool res = a >= b`	
`<=`	Less than or equal to	`bool res = a <= b`	

C++ Relational Operators

Operator	Meaning	Example	Value of res for a = 3 and b = 5
`==`	Is equal to	`bool res = a == b`	`false`
`!=`	Not equal to	`bool res = a != b`	`true`
`>`	Greater than	`bool res = a > b`	`false`
`<`	Less than	`bool res = a < b`	`true`
`>=`	Greater than or equal to	`bool res = a >= b`	
`<=`	Less than or equal to	`bool res = a <= b`	

C++ Relational Operators

Operator	Meaning	Example	Value of res for a = 3 and b = 5
`==`	Is equal to	`bool res = a == b`	`false`
`!=`	Not equal to	`bool res = a != b`	`true`
`>`	Greater than	`bool res = a > b`	`false`
`<`	Less than	`bool res = a < b`	`true`
`>=`	Greater than or equal to	`bool res = a >= b`	`false`
`<=`	Less than or equal to	`bool res = a <= b`	

C++ Relational Operators

Operator	Meaning	Example	Value of res for a = 3 and b = 5
`==`	Is equal to	`bool res = a == b`	`false`
`!=`	Not equal to	`bool res = a != b`	`true`
`>`	Greater than	`bool res = a > b`	`false`
`<`	Less than	`bool res = a < b`	`true`
`>=`	Greater than or equal to	`bool res = a >= b`	`false`
`<=`	Less than or equal to	`bool res = a <= b`	`true`

C++ Logical Operators

Operator	Meaning	Example	Value of res for a = 3 and b = -5
&&	Logical AND *True if all the operands are true.*	`bool res = (a > 0) && (b > 0)`	
||	Logical OR *True if at least one of the operands is true.*	`bool res = (a > 0) || (b > 0)`	
!	Logical NOT *True only if the operand is false.*	`bool res = !(a > 0)`	

C++ Logical Operators

<table>
<tr><th>Operator</th><th>Meaning</th><th>Example</th><th>Value of res for
a = 3 and b = -5</th></tr>
<tr><td>&&</td><td>Logical AND
<em>True if all the operands are true.</em></td><td><code>bool res = (a > 0) && (b > 0)</code></td><td><code>false</code></td></tr>
<tr><td>||</td><td>Logical OR
<em>True if at least one of the operands is true.</em></td><td><code>bool res = (a > 0) || (b > 0)</code></td><td></td></tr>
<tr><td>!</td><td>Logical NOT
<em>True only if the operand is false.</em></td><td><code>bool res = !(a > 0)</code></td><td></td></tr>
</table>

C++ Logical Operators

Operator	Meaning	Example	Value of res for a = 3 and b = -5
`&&`	Logical AND *True if all the operands are true.*	`bool res = (a > 0) && (b > 0)`	`false`
`||`	Logical OR *True if at least one of the operands is true.*	`bool res = (a > 0) || (b > 0)`	`true`
`!`	Logical NOT *True only if the operand is false.*	`bool res = !(a > 0)`	

C++ Logical Operators

<table>
<tr><th>Operator</th><th>Meaning</th><th>Example</th><th>Value of res for
a = 3 and b = -5</th></tr>
<tr><td>&&</td><td>Logical AND
<i>True if all the operands are true.</i></td><td><code>bool res = (a > 0) && (b > 0)</code></td><td><code>false</code></td></tr>
<tr><td>||</td><td>Logical OR
<i>True if at least one of the operands is true.</i></td><td><code>bool res = (a > 0) || (b > 0)</code></td><td><code>true</code></td></tr>
<tr><td>!</td><td>Logical NOT
<i>True only if the operand is false.</i></td><td><code>bool res = !(a > 0)</code></td><td><code>false</code></td></tr>
</table>

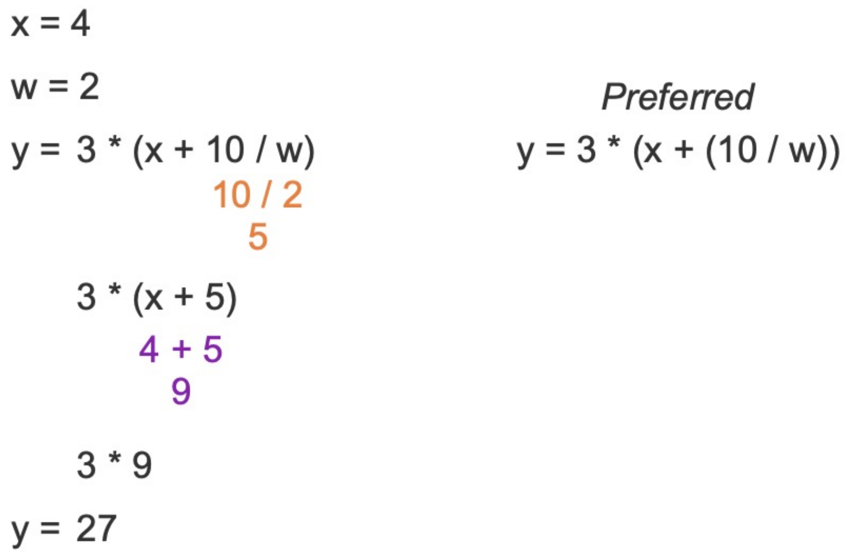
C++ Operator Precedence

Precedence	Operator	Description	Associativity
1	a++, a--	Post-increment, post-decrement	Left-to-right
2	++a, --a, sizeof	Pre-increment, pre-decrement, size of	Right-to-left
3	a * b, a / b, a % b	Multiplication, division, modulus	Left-to-right
4	a + b, a - b	Addition, subtraction	
5	< <= > >=	Relational operator	
6	== !=	Equality operator	
7	&&	Logical AND	
8	||	Logical OR	

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

Arithmetic Expressions

- Use of *parentheses*
- Brackets [] or braces {} may NOT be used.
- C++ operator precedence



```
x = 4
w = 2                                      Preferred
y = 3 * (x + 10 / w)               y = 3 * (x + (10 / w))
              10 / 2
                5
     3 * (x + 5)
          4 + 5
            9
     3 * 9
y = 27
```

Image source: ZyBooks – Chapter 2 (2.7.3)

Output Tracing

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

int main()
{
        int a = 2;
        cout << a + a * a - a << endl;
        cout << a * ++a - a << endl;
        cout << a << endl;
        return 0;
}
```

```
??
```

Output Tracing

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

int main()
{
        int a = 2;
        cout << a + a * a - a << endl;
        cout << a * ++a - a << endl;
        cout << a << endl;
        return 0;
}
```

```
4
6
3
```

Character Data Type (1/2)

- Character (`char`)
  - Size: 1 byte or 8 bits
  - *Signed* range: -128 to +128
  - *Unsigned* range: 0 to 255
- Created by enclosing a single character inside **single** quotation marks

```
char LetterGrade = ‘A’;
```

Further Reading (Optional): https://docs.microsoft.com/en-us/cpp/cpp/fundamental-types-cpp?view=msvc-160

Character Data Type (2/2)

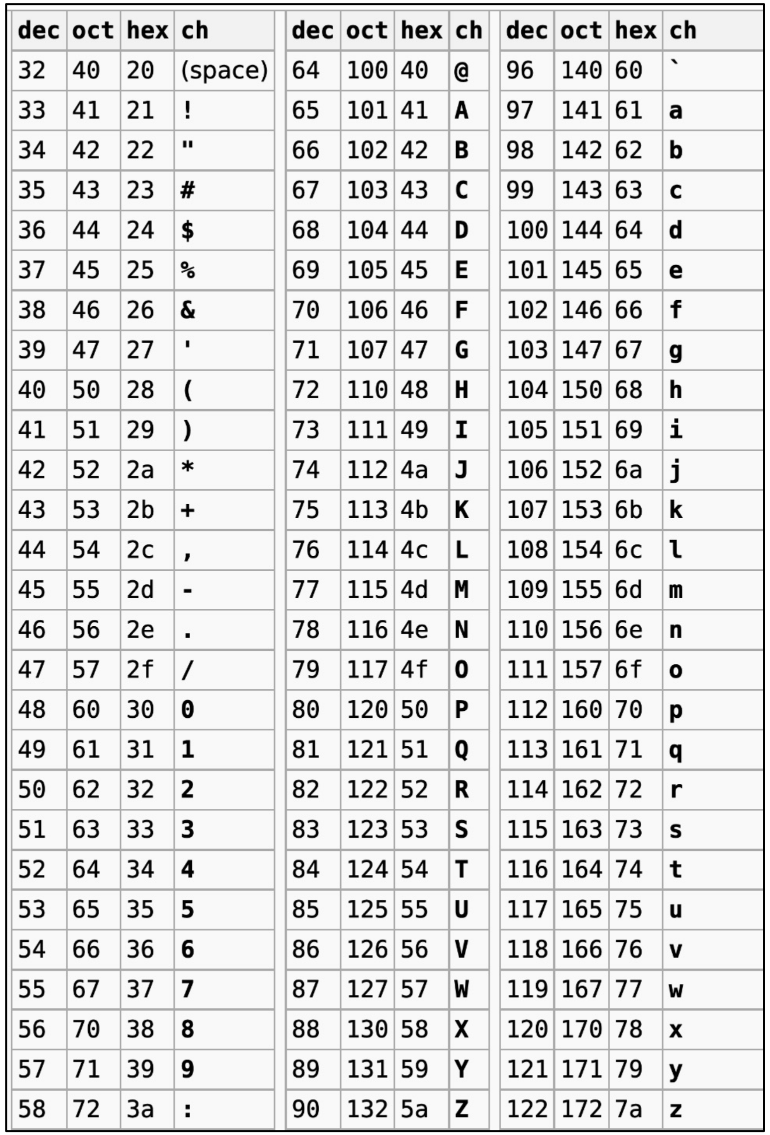
- Internally stored as numbers

```
LetterGrade = ‘A’;
```

66

```
LetterGrade = ‘B’;
```

67



dec	oct	hex	ch	dec	oct	hex	ch	dec	oct	hex	ch
32	40	20	(space)	64	100	40	@	96	140	60	`
33	41	21	!	65	101	41	A	97	141	61	a
34	42	22	"	66	102	42	B	98	142	62	b
35	43	23	#	67	103	43	C	99	143	63	c
36	44	24	$	68	104	44	D	100	144	64	d
37	45	25	%	69	105	45	E	101	145	65	e
38	46	26	&	70	106	46	F	102	146	66	f
39	47	27	'	71	107	47	G	103	147	67	g
40	50	28	(	72	110	48	H	104	150	68	h
41	51	29	)	73	111	49	I	105	151	69	i
42	52	2a	*	74	112	4a	J	106	152	6a	j
43	53	2b	+	75	113	4b	K	107	153	6b	k
44	54	2c	,	76	114	4c	L	108	154	6c	l
45	55	2d	-	77	115	4d	M	109	155	6d	m
46	56	2e	.	78	116	4e	N	110	156	6e	n
47	57	2f	/	79	117	4f	O	111	157	6f	o
48	60	30	0	80	120	50	P	112	160	70	p
49	61	31	1	81	121	51	Q	113	161	71	q
50	62	32	2	82	122	52	R	114	162	72	r
51	63	33	3	83	123	53	S	115	163	73	s
52	64	34	4	84	124	54	T	116	164	74	t
53	65	35	5	85	125	55	U	117	165	75	u
54	66	36	6	86	126	56	V	118	166	76	v
55	67	37	7	87	127	57	W	119	167	77	w
56	70	38	8	88	130	58	X	120	170	78	x
57	71	39	9	89	131	59	Y	121	171	79	y
58	72	3a	:	90	132	5a	Z	122	172	7a	z

Further Explanation: https://www.youtube.com/watch?v=jjqgP9dpD1k&list=PLhQjrBD2T381L3iZyDTxRwOBuUt6m1FnW&index=1&t=793s

Type Conversion

- Implicit conversion
  - Automatically performed when a value is copied to a compatible type

```
short sMyVar = 20;
int iMyVar;
iMyVar = sMyVar;
```

Further Reading (Optional): https://www.cplusplus.com/doc/tutorial/typecasting/

Type Conversion

- Implicit conversion
  - Automatically performed when a value is copied to a compatible type

```
short sMyVar = 20;
int iMyVar;
iMyVar = sMyVar;
```

```
int iMyVar = 20;
double dMyVar;
dMyVar = iMyVar;
cout << “int: ” << iMyVar << “ , double: ” << dMyVar << endl;
```

```
int: 11, double: 201
```

Type Conversion

- Implicit conversion
  - Automatically performed when a value is copied to a compatible type

```
short sMyVar = 20;
int iMyVar;
iMyVar = sMyVar;
```

```
int iMyVar = 20;
double dMyVar;
dMyVar = iMyVar;
cout << “int: ” << iMyVar << “ , double: ” << dMyVar << endl;
```

```
int: 11, double: 201
```

```
int iMyVar;
double dMyVar = 20.5;
iMyVar = dMyVar;
cout << “int: ” << iMyVar << “ , double: ” << dMyVar << endl;
```

```
int: 20, double: 20.5
```

Type Conversion

```
int iMyVar;
double dMyVar = 20.5;
iMyVar = dMyVar;
cout << “int: ” << iMyVar << “ , double: ” << dMyVar << endl;
```

```
int: 20, double: 20.5
```

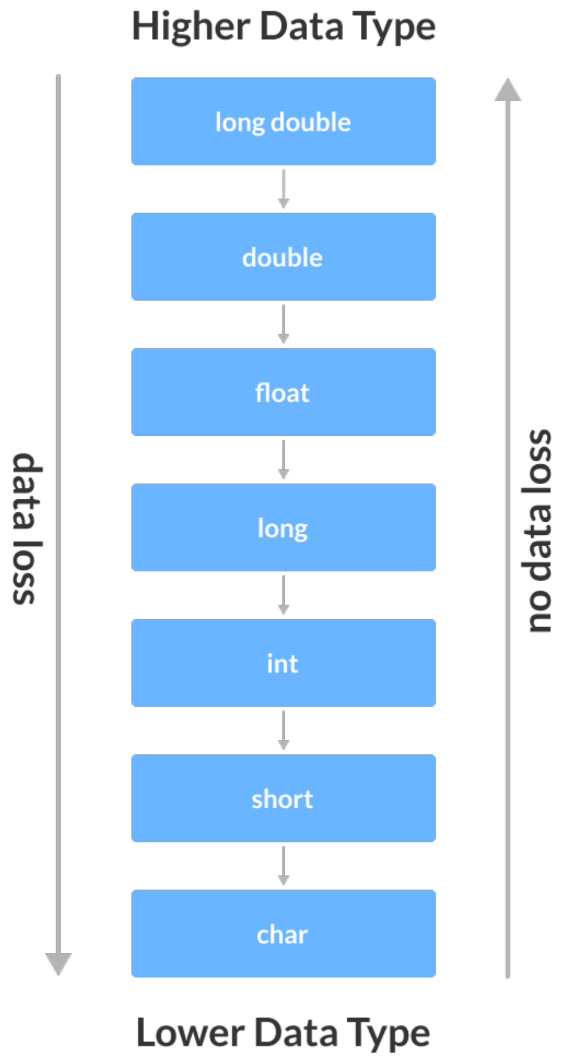


Image Source: https://www.programiz.com/cpp-programming/type-conversion

Type Conversion

- Explicit Conversion
  - Manually change data from one type to another

```
(data_type)expression;
```

```
int iMyVar = 20;
double dMyVar;
dMyVar = (double)iMyVar;
```

```
data_type(expression);
```

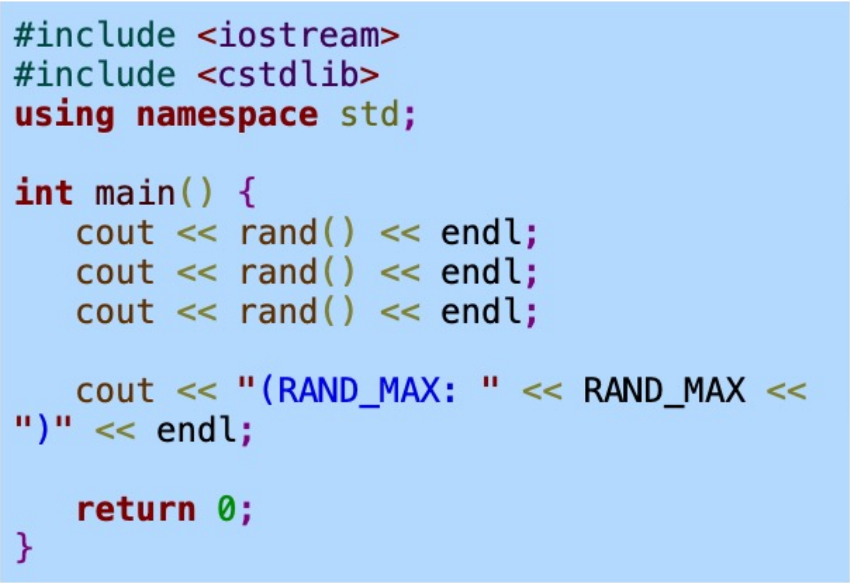
```
int iMyVar = 20;
double dMyVar;
dMyVar = double(iMyVar);
```

```
static_cast<data_type>(expression);
```

```
int iMyVar = 20;
double dMyVar;
dMyVar = static_cast<double>iMyVar;
```

Random Number (`rand`)

- A pseudo-random integral number in the range between `0` and `RAND_MAX`.
- `RAND_MAX` (a constant defined in `<cstdlib>`) is a machine dependent value, but is at least `32,767` – which is `(2`^{16}`-1)/2`.

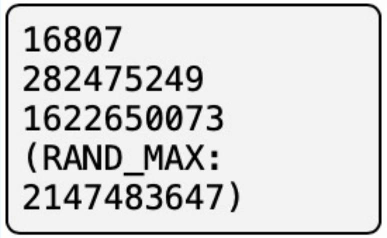


```
#include <iostream>
#include <cstdlib>
using namespace std;

int main() {
   cout << rand() << endl;
   cout << rand() << endl;
   cout << rand() << endl;

   cout << "(RAND_MAX: " << RAND_MAX <<
")" << endl;

   return 0;
}
```



```
16807
282475249
1622650073
(RAND_MAX:
2147483647)
```