1-BASICS OF C++

STRUCTURE OF A PROGRAM

hash
#include <iostream> header
The main function is called when the program is run.

indicate = belirtmek, göstermek

std::cout << “Hello World!” ; statement

std::cout **st**andar**d** **c**haracter **out**put

discard = gözden çıkarmak
unqualified = vasıfsız, nitelenmemiş

Variables And Types

retaining = alıkoyma, tespit etme, tutma
come up with = bulmak, ileri sürmek, düşünmek

A valid identifier is a sequence of one or more letters, digits, or underscore characters (_).

occupy = meşgul etmek, işgal etmek

Here is the complete list of fundamental types in C++:

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)**	

the part in italics is optional. I.e., *signed* short *int* can be abbreviated as signed short, short int, or simply short; they all identify the same fundamental type.

Size	Unique representable values	Notes
8-bit	256	$= 2^8$
16-bit	65 536	$= 2^{16}$
32-bit	4 294 967 296	$= 2^{32}$ (~4 billion)
64-bit	18 446 744 073 709 551 616	$= 2^{64}$ (~18 billion billion)

modest = alçak gönüllü, gösterişsiz
precision = kesinlik, duyarlık, tamlık
exponent = kuvvet, üs, sembol, yorumcu
respectively = sırasıyla

void which identifies the lack of type
straightforward = apaçık, özü sözü bir
proposed = önerilen
reminiscent = hatırlatan, andıran

In C++, there are three ways to initialize variables.
int x = 0 ; (C-like initialization)
int x (0) ; (Constructor initalization)
int x = { 0 } ; (Uniform initialization)

figure out = çözmek, bir şeyin nedenini kestirmek
suffice = yetmek, yeterli olmak
deduction = kesinti, tümdengelim, sonuç çıkarma

```
1 int foo = 0;
2 auto bar = foo;  // the same as: int bar = foo;
```

Here, **bar** is declared as having an **auto** type; it uses the type of **foo,** which is int.

```
1 int foo = 0;
2 decltype(foo) bar;  // the same as: int bar;
```

bar is declared as having the same type as **foo.**

to be used either when the type cannot be obtained by other means or when using it improves code readability.

handle = idare etmek, üstesinden gelmek
mere = yalnız
merely = yalnızca

```
1 string mystring = "This is a string";
2 string mystring ("This is a string");
3 string mystring {"This is a string"};
```

The **string** class is a **compound type.** They are used in the same way as fundamental types: the same syntax is used to declare variables and to initialize them.

endl manipulator (**end**s the **l**ine), printing a newline character and flushing the stream.

CONSTANTS
literal = tam, kelimesi kelimesine
Literals are used to express particular values within the source code of a program.

Literal constants can be classified into: integer, floating-point, characters, strings, Boolean, pointers, user-defined literals.

decimal = onluk, ondalık sayı
octal = sekizlik
hexadecimal = on altılık

75 //decimal
0113 //octal (preceded with a 0)
0x4b //hexadecimal (preceded with a 0x)
All of these represents the same number, 75.

75 // int
75u // unsigned int
75l // long
75ul // unsigned long
75lu // unsigned long

e character expresses “by ten at the Xth height” where X is an integer value that follows the **e** character.

6.02e23 // 6.02 x 10 ^ 23
1.6e-19 // 1.6 x 10 ^ -19

Character and string literals are enclosed in quotes:
‘z’ (single character literals) ‘p’ “ Hello World! “ (string literals)

Single Character Escape Codes:

Escape code	Description
\n	newline
\r	carriage return
\t	tab
\v	vertical tab
\b	backspace
\f	form feed (page feed)
\a	alert (beep)
\'	single quote (')
\"	double quote (")
\?	question mark (?)
\\	backslash (\)

For an octal value, the backslash is followed directly by the digits; while for hexadecimal, an x character is inserted between the backslash and the hexadecimal digits themselves (for example: \x20 or \x4A).

internally = içten, dahilen

The standard ASCII table defines 128 character codes (from 0 to 127), of which, the first 32 are control codes (non-printable), and the remaining 96 character codes are representable characters:

*	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
0	NUL	SOH	STX	ETX	EOT	ENQ	ACK	BEL	BS	TAB	LF	VT	FF	CR	SO	SI
1	DLE	DC1	DC2	DC3	DC4	NAK	SYN	ETB	CAN	EM	SUB	ESC	FS	GS	RS	US
2		!	"	#	$	%	&	'	(	)	*	+	,	-	.	/
3	0	1	2	3	4	5	6	7	8	9	:	;	<	=	>	?
4	@	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O
5	P	Q	R	S	T	U	V	W	X	Y	Z	[	\	]	^	_
6	`	a	b	c	d	e	f	g	h	i	j	k	l	m	n	o
7	p	q	r	s	t	u	v	w	x	y	z	{	|	}	~	

The "A" character is located at the 4th row and the 1st column, for that it would be represented in hexadecimal as 0x41 (65).

concatenate = sıralamak, ard arda bağlamak

Prefix	Character type
u	char16_t
U	char32_t
L	wchar_t

Prefix	Description
u8	The string literal is encoded in the executable using UTF-8
R	The string literal is a raw string

delimited = sınırlandırılmış

```
1 bool foo = true;
2 bool bar = false;
3 int* p = nullptr;
```

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

const double pi = 3.14159;
const char newline = '\n';

int main ()
{
  double r=5.0;                   // radius
  double circle;

  circle = 2 * pi * r;
  cout << circle;
  cout << newline;
}
```

```
31.4159
```

Preprocessor definitions (# define)
Another mechanism to name constant values.

occurrence = oluşum, olay, meydana gelme
validity = geçerlilik, doğruluk

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

#define PI 3.14159
#define NEWLINE '\n'

int main ()
{
  double r=5.0;                   // radius
  double circle;

  circle = 2 * PI * r;
  cout << circle;
  cout << NEWLINE;

}
```

```
31.4159
```

OPERATORS
The assignment operation (=) always takes place from right to left.

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

y = 2+x;

x=y=z=5;
% (modulo operator) gives the remainder of a division two values. x = 11 % 3; (x=2)

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);`

++x; x+=1; x=x+1; x++;
peculiarity = acayiplik, alışılmamışlık
in the case that = olması halinde

Example 1	Example 2

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

operator	description
==	Equal to
!=	Not equal to
<	Less than
>	Greater than
<=	Less than or equal to
>=	Greater than or equal to

int a = 5;
a == 5 //evaluates to true

! is the C++ operator for the Boolean operation NOT. It returns the opposite Boolean value of evaluating its operand.

When using the logical operators (&& ll), C++ only evaluates what is necessary from left to right to come up with the combined relational result, ignoring the rest.
(short circuit evaluation)

operator	short-circuit
&&	if the left-hand side expression is `false`, the combined result is `false` (the right-hand side expression is never evaluated).
||	if the left-hand side expression is `true`, the combined result is `true` (the right-hand side expression is never evaluated).

if ((i<10) && (++i < n)) //the condition increments i
The combined conditional expression would increase **i** by one, but only if the condition on the left of && is **true,** because otherwise, the condition on the right-hand side (++i < n) is never evaluated.

Conditional Ternary Operator (?)

```
7==5 ? 4 : 3      // evaluates to 3, since 7 is not equal to 5.
7==5+2 ? 4 : 3    // evaluates to 4, since 7 is equal to 5+2.
5>3 ? a : b       // evaluates to the value of a, since 5 is greater than 3.
a>b ? a : b       // evaluates to whichever is greater, a or b.
```

```
int a,b,c;
   a=2;
   //b=7;
   //c = (a>b)? a:b;
   a = (b=3, b+2);
   cout<<a<<' \n ';
```

in favor of = lehine, -den yana
inherited = miras kalan, kalıtsal

a = (b=3, b+2); // a = 5;

Type casting operators allow to convert a value of a given type to another type.
int i; float f = 3.14; i = (int) f;
The previous code converts the floating-point number 3.14 to an integer value (3), the remainder is lost.
i = int (f); // is valid also for type casting

sizeof accepts one parameter which can be either a type or a variable, and returns the size in bytes of that type or object.
x = **sizeof** (char);
The value returned by **sizeof** is a compile-time constant, so it is always determined before program execution.

precedence = öncelik, üstünlük, öncelik sırası
x = 5 + 7 % 2; // x=6

x = 5 (7%2); // x=6
x = (5+7) % 2; // x=0

From greatest to smallest priority, C++ operators are evaluated in the following order:

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

BASIC INPUT / OUTPUT

entity = varlık, mevcudiyet
precede = -den üstün olmak
literal = kalıp deyim, hazır bilgi, harfi harfine
zip code = posta kodu
incur = -den kaynaklanmak
straightforward = apaçık, basit
drawback = güçlük, eksiklik

cin extraction always considers spaces as terminating the value being extracted, and thus extracting a string means to always extract a single word, not a phrase or an entire sentence.

```
string mystring;
getline ( cin , mystring );
```

vice versa = tam tersi

<sstream> (stringstream), to convert strings to numerical values and vice versa. In order to extract an integer from a string we can write.

```
string      mystring ( “1204” );
int                 myint;
stringstream ( mystring ) >> myint;
```

———————————————

OEM - Original Equipment Manufacturer
immense - harika, çok büyük (extremely large or big)
vendors - satıcılar
prone - eğilimli (error prone)

0x16	(SYN)	22(decimal)
0x07	(ACK)	7(decimal)

indented - girintili
leginle - okunaklı
distinct - belirgin, şüphesiz
halved - ikiye bölünen
integral - tümlev, bütünleyen şey
reminiscent - andıran