哈爾濱工業大學(深圳)
HARBIN INSTITUTE OF TECHNOLOGY, SHENZHEN

高级语言程序设计
High-level Language Programming

Lecture 2 Basic syntax of C++

Yitian Shao (shaoyitian@hit.edu.cn)
School of Computer Science and Technology

Basics syntax of C++

Course Overview

- Variable
  - Identifiers
  - Data types
  - Literals
- Variable properties
  - Name, type, and value
  - Memory storage
- Variable definition and assignment

Try it yourself on https://www.onlinegdb.com

Let’s start with a “hello world” example

Try it yourself on https://www.onlinegdb.com

Let's start with a "hello world" example

ONLINE CPP

C++

main.cpp

```
1  #include<iostream>
2
3  int main()
4  {
5      int my_number = 2024;
6
7      char my_text[20] = "Hello world ";
8
9      std::cout << my_text << my_number;
10
11     return 0;
12 }
13
14
```

Ln: 14, Col: 1

Run | Share | Command Line Arguments

```
Hello world 2024

** Process exited - Return Code: 0 **
```

We create two variables (what we learn today)

Try it yourself on https://www.onlinegdb.com

C++ hello world example

ONLINE CPP

main.cpp

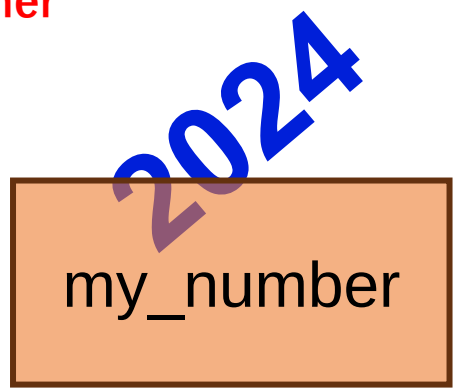
```cpp
#include<iostream>

int main()
{
    int my_number = 2024;

    char my_text[20] = "Hello world ";

    std::cout << my_text << my_number;

    return 0;
}
```

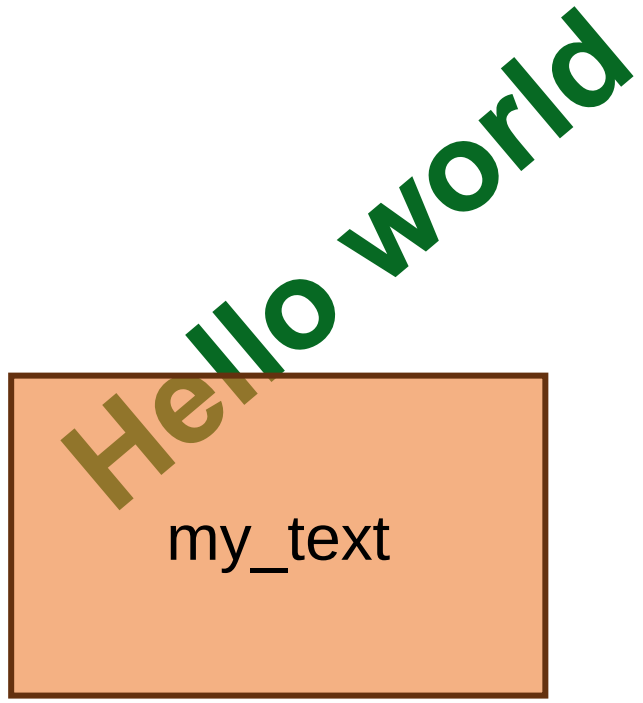
Ln: 14, Col: 1

Run | Share | Command Line Arguments

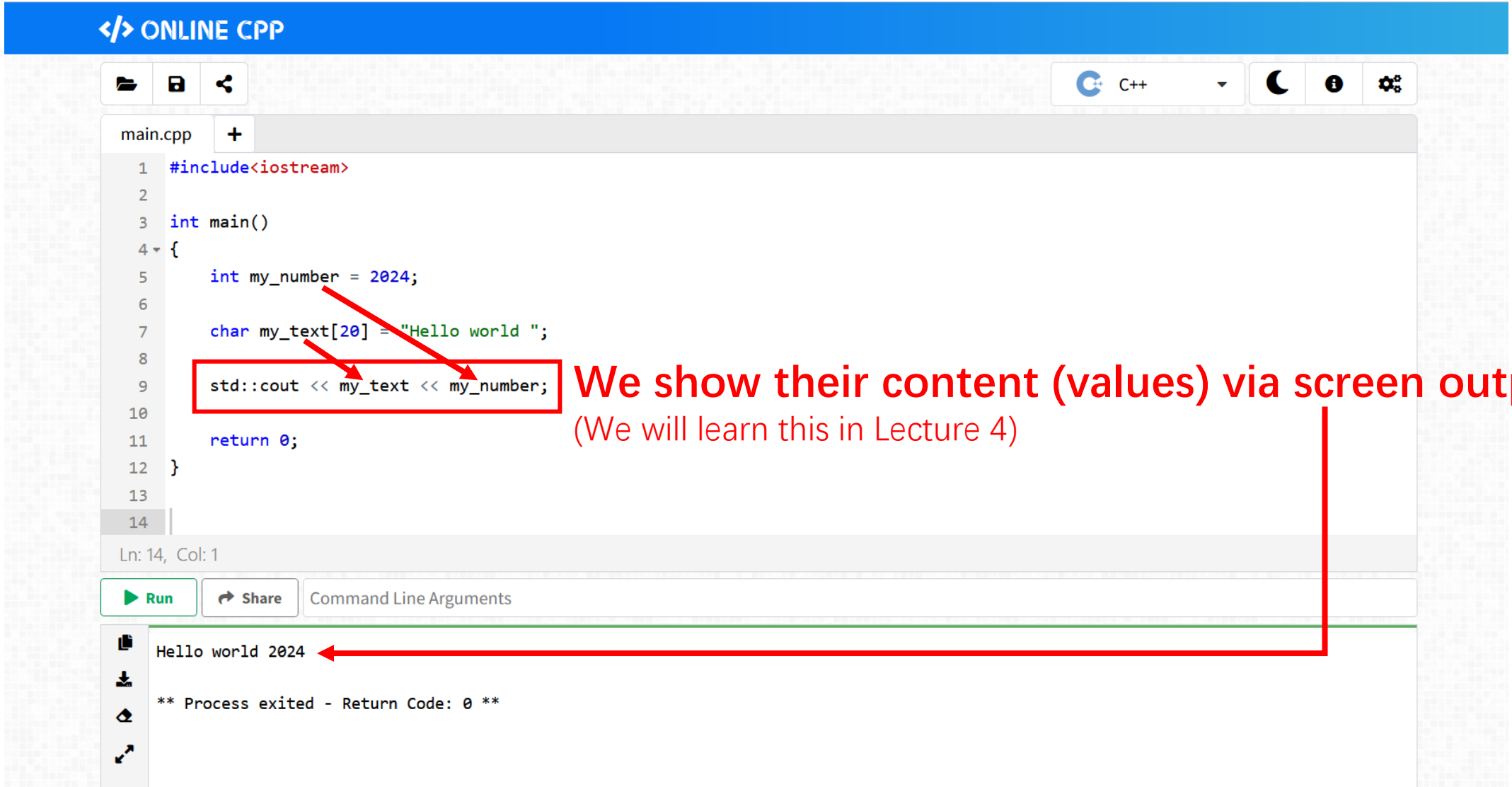
```
Hello world 2024

** Process exited - Return Code: 0 **
```

Like a "Container"





Let's start with a "hello world" example



We show their content (values) via screen output

(We will learn this in Lecture 4)

Try it yourself on https://www.onlinegdb.com

C++ hello world example

ONLINE CPP

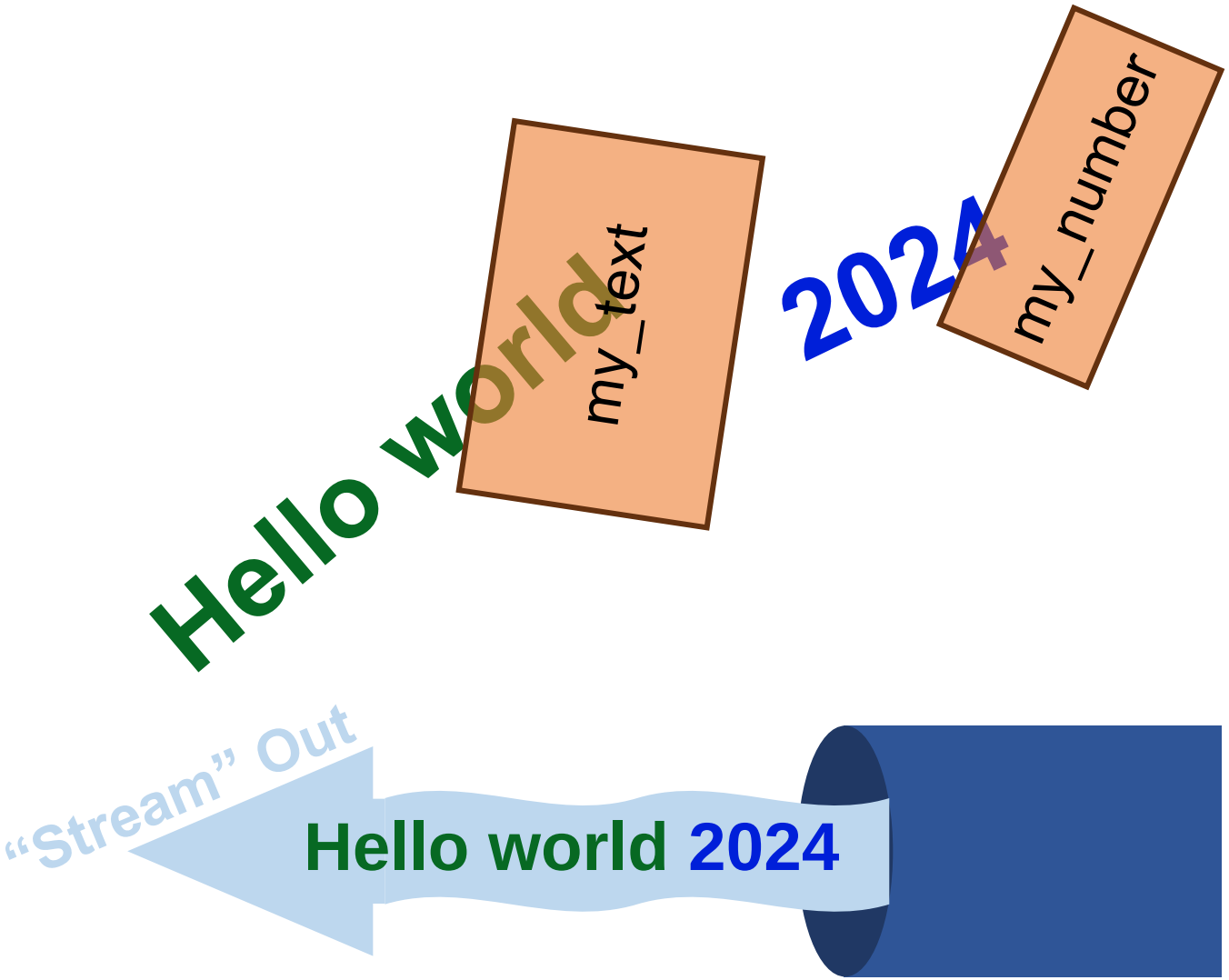
main.cpp

```
1  #include<iostream>
2
3  int main()
4  {
5      int my_number = 2024;
6
7      char my_text[20] = "Hello world ";
8
9      std::cout << my_text << my_number;
10
11     return 0;
12 }
13
14
```

Ln: 14, Col: 1

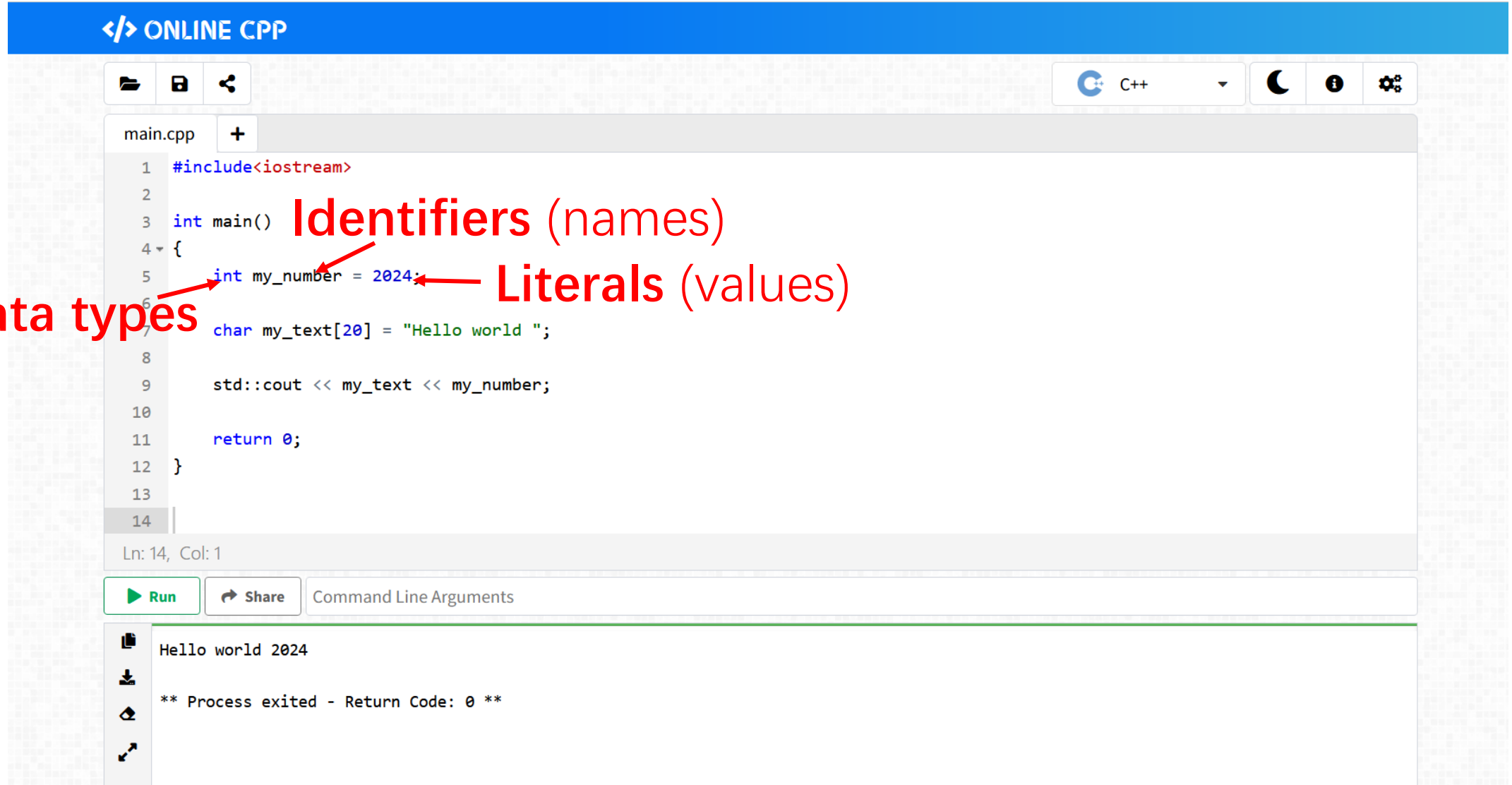
Run | Share | Command Line Arguments

```
Hello world 2024

** Process exited - Return Code: 0 **
```



Try it yourself on https://www.onlinegdb.com

C++ hello world example



Try it yourself on https://www.onlinegdb.com

C++ hello world example

ONLINE CPP

C++

main.cpp

```
1  #include<iostream>
2
3  int main()
4  {
5      int my_number = 2024;
6
7      char my_text[20] = "Hello world ";
8
9      std::cout << my_text << my_number;
10
11     return 0;
12 }
13
14
```

Ln: 14, Col: 1

Run | Share | Command Line Arguments

```
Hello world 2024

** Process exited - Return Code: 0 **
```

Identifiers used to name entities, such as constants, variables, or functions

Try it yourself on https://www.onlinegdb.com

Identifiers

- Valid identifier is sequence of one or more **letters, digits,** and **underscore characters** that does **not begin with a digit**

- Identifiers that **begin with underscore** or **contain double underscores** are reserved for use by C++ implementation and should be avoided whenever possible

- Example of valid identifiers:

  - `event_counter`
  - `eventCounter`
  - `sqrt_2`
  - `f_o_o_b_a_r_4_2`

Identifiers

- Identifiers are **case sensitive**

  For example, the following are distinct identifiers:

  `counter` `cOuNtEr`

- Identifiers cannot be any of reserved keywords

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

*Note: context sensitive

Data types

ONLINE CPP

C++

main.cpp

```
1  #include<iostream>
2
3  int main()
4  {
5      int my_number = 2024;
6
7      char my_text[20] = "Hello world ";
8
9      std::cout << my_text << my_number;
10
11     return 0;
12 }
13
14
```

Ln: 14, Col: 1

Run | Share | Command Line Arguments

```
Hello world 2024

** Process exited - Return Code: 0 **
```

Data types

determine the **allocated memory space**,
the **storage format of data**,
the **valid range of values**, and
the **types of operations it can participate in**

Basic data types

- Boolean type **`bool`**
- Integer type (signed and unsigned)

```
signed char              unsigned char
signed short int         unsigned short int
signed int               unsigned int
signed long int          unsigned long int
signed long long int     unsigned long long int
```

- Floating-point types **`float`** **`double`** **`long double`**
- Character types **`char`**

We will learn more details about the basic data types later in this lecture…

- Void type (incomplete/valueless) **`void`**
- Pointer type **`int*`** **`char*`**
- Array type

```
int ages[10]    char my_text[20]
```

Literals

ONLINE CPP

C++

main.cpp

```cpp
#include<iostream>

int main()
{
    int my_number = 2024;

    char my_text[20] = "Hello world ";

    std::cout << my_text << my_number;

    return 0;
}
```

Literals

Ln: 14, Col: 1

Run | Share | Command Line Arguments

```
Hello world 2024

** Process exited - Return Code: 0 **
```

Literals

- Literal (literal constant) is value written exactly as it is meant to be interpreted; it is a fixed value that the program may not alter

- Examples of literals:

```
"Hello, world"
"Bjarne"
'a'
'A'
123
123U
1'000'000'000
3.1415
1.0L
```

main.cpp +

```cpp
#include<iostream>

int main()
{
    int my_number = 2024;

    char my_text[20] = "Hello world ";

    std::cout << my_text << my_number;

    return 0;
}
```

Literals

- Boolean literals: **`true`** **`false`**

- Pointer literal: **`nullptr`**

Integer literal types

- **Prefixes** which indicates the **base**. For example, **0x**10 indicates the value 16 in hexadecimal having prefix **0x**.
    - **Decimal-literal** (base 10):- **a non-zero decimal digit** followed by zero or more decimal digits(0, 1, 2, 3, 4, 5, 6, 7, 8, 9). For example, 56, 78.
    - **Hex-literal** (base 16):- **0x or 0X** followed by one or more hexadecimal digits(0, 1, 2, 3, 4, 5, 6, 7, 8, 9, a, A, b, B, c, C, d, D, e, E, f, F). For example, 0x23A, 0Xb4C, 0xFEB.

Integer literal types

- **Prefixes** which indicates the **base**. For example, **0x**10 indicates the value 16 in hexadecimal having prefix **0x**.
  - **Decimal-literal** (base 10):- **a non-zero decimal digit** followed by zero or more decimal digits(0, 1, 2, 3, 4, 5, 6, 7, 8, 9). For example, 56, 78.
  - **Hex-literal** (base 16):- **0x or 0X** followed by one or more hexadecimal digits(0, 1, 2, 3, 4, 5, 6, 7, 8, 9, a, A, b, B, c, C, d, D, e, E, f, F). For example, 0x23A, 0Xb4C, 0xFEB.

Conversion from Hexadecimal to decimal:

0x10 = 1 * 16 + 0 = 16 **(decimal)**

0x23A = 2 * 16^2 + 3 * 16 + A = 512 + 48 + 10 = 570 **(decimal)**

0xFEA = F * 16^2 + E * 16 + B = 3840 + 224 + 11 = 4075 **(decimal)**

A	10
B	11
C	12
D	13
E	14
F	15

Integer literal types

- **Prefixes** which indicates the **base**. For example, **0x**10 indicates the value 16 in hexadecimal having prefix **0x**.
  - **Decimal-literal** (base 10):- **a non-zero decimal digit** followed by zero or more decimal digits(0, 1, 2, 3, 4, 5, 6, 7, 8, 9). For example, 56, 78.
  - **Hex-literal** (base 16):- **0x or 0X** followed by one or more hexadecimal digits(0, 1, 2, 3, 4, 5, 6, 7, 8, 9, a, A, b, B, c, C, d, D, e, E, f, F). For example, 0x23A, 0Xb4C, 0xFEA.
  - **Binary-literal** (base 2):- **0b or 0B** followed by one or more binary digits(0, 1). For example, 0b101, 0B111.
  - **Octal-literal** (base 8):- **a 0** followed by zero or more octal digits(0, 1, 2, 3, 4, 5, 6, 7). For example, 045, 076, 06210.

  Question: what are the values stored inside the following variables?

```
int d = 42;
int o = 052;
int x = 0x2a;
int X = 0X2A;
int b = 0b101010;
```

Integer literal types

- **Prefixes** which indicates the **base**. For example, **0x**10 indicates the value 16 in hexadecimal having prefix **0x**.
  - **Decimal-literal** (base 10):- **a non-zero decimal digit** followed by zero or more decimal digits(0, 1, 2, 3, 4, 5, 6, 7, 8, 9). For example, 56, 78.
  - **Hex-literal** (base 16):- **0x or 0X** followed by one or more hexadecimal digits(0, 1, 2, 3, 4, 5, 6, 7, 8, 9, a, A, b, B, c, C, d, D, e, E, f, F). For example, 0x23A, 0Xb4C, 0xFEA.
  - **Binary-literal** (base 2):- **0b or 0B** followed by one or more binary digits(0, 1). For example, 0b101, 0B111.
  - **Octal-literal** (base 8):- **a zero** followed by zero or more octal digits(0, 1, 2, 3, 4, 5, 6, 7). For example, 045, 076, 06210.

  Question: what are the values stored inside the following variables?

```
int d = 42;
int o = 052;
int x = 0x2a;
int X = 0X2A;
int b = 0b101010;
```

$= 5 * 8 + 2 = 42$

$= 2 * 16 + A = 42$

$= 1 * 2^5 + 1 * 2^3 + 1 * 2^1 = 32 + 8 + 2 = 42$

Integer literal types

- **Suffixes** which indicates the **data type**. For example, 12345678901234**LL** indicates the value 12345678901234 as an long long integer having suffix **LL**.
  - **int**:- No suffix are required because integer constant are by default assigned as int data type.
  - **unsigned int**: character **u or U** at the end of integer constant.
  - **long int**: character **l or L** at the end of integer constant.
  - **unsigned long int**: character **ul or UL** at the end of integer constant.
  - **long long int**: character **ll or LL** at the end of integer constant.
  - **unsigned long long int**: character **ull or ULL** at the end of integer constant.

Floating-point literal types

Suffix	Type
None	**double**
f or F	**float**
l or L	**long double**

```
1.414
1.25e-8
```
1.25×10^{-8}

```
1.414f
1.25e-8f
```
1.25×10^{-8}

Suffix

```
1.5L
1.25e-20L
```
1.25×10^{-20}

Expressions with **e**xponents

$\mathbf{s \times r^{j}}$

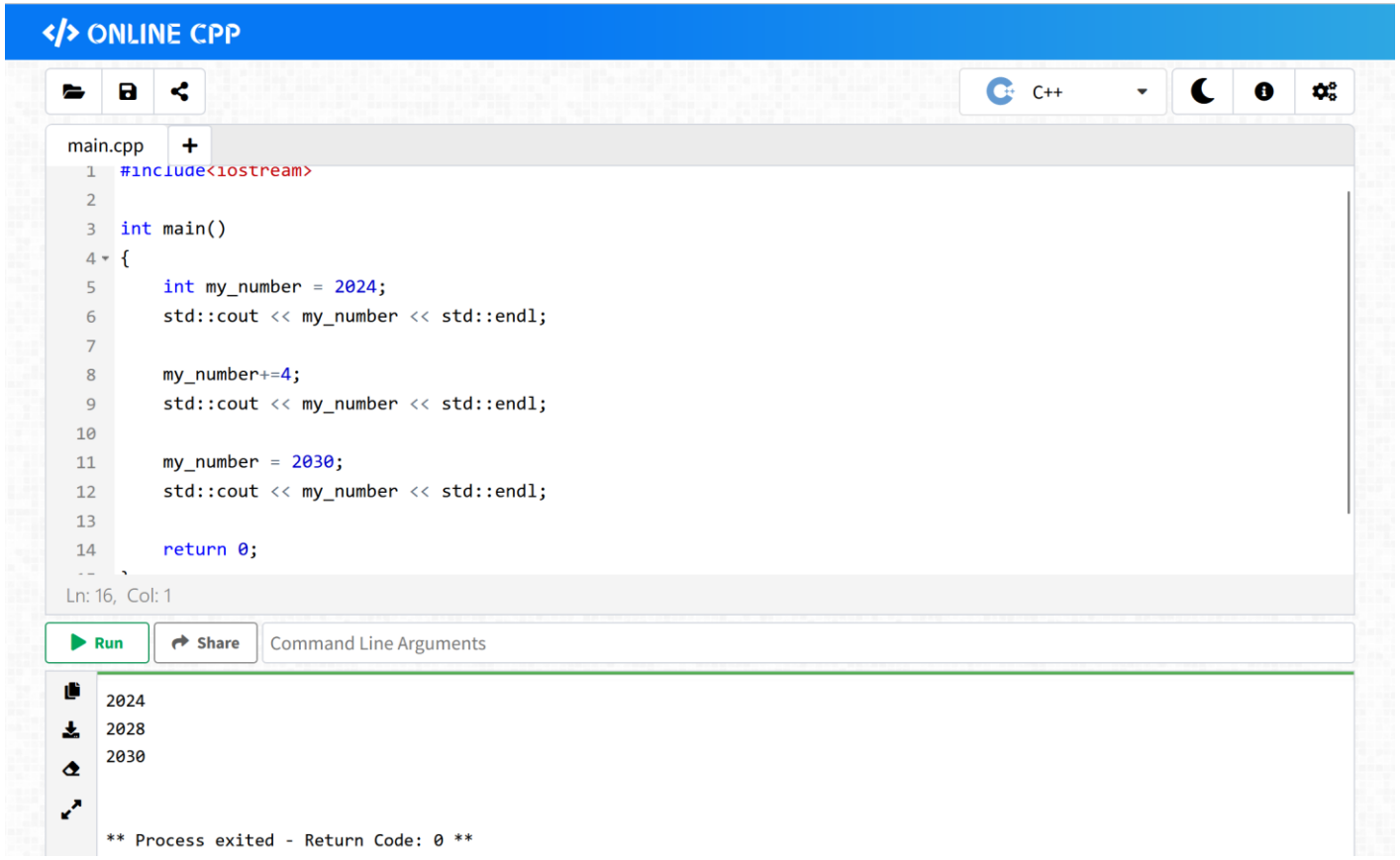
Significand s

Exponent j (how many times to use the number in a multiplication)

Radix r (base)

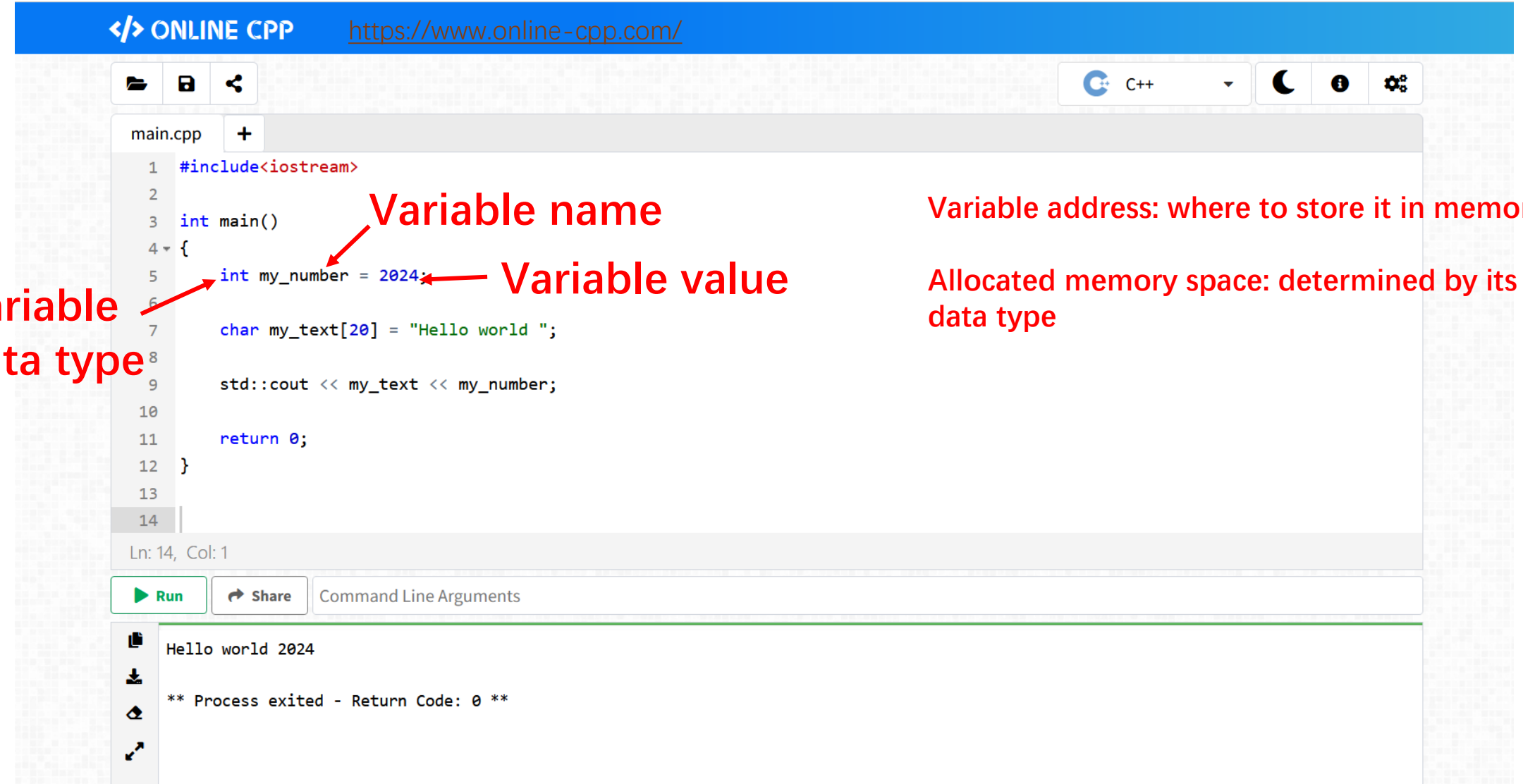
Constants and variables

- Constant: a fixed value that the program may not alter
- Variable: the value stored inside **can vary** during program execution



Variables

ONLINE CPP https://www.online-cpp.com/



Variable name

Variable value

Variable data type

Variable address: where to store it in memory

Allocated memory space: determined by its data type

https://www.onlinegdb.com/

Data types determine allocated memory space

ONLINE CPP

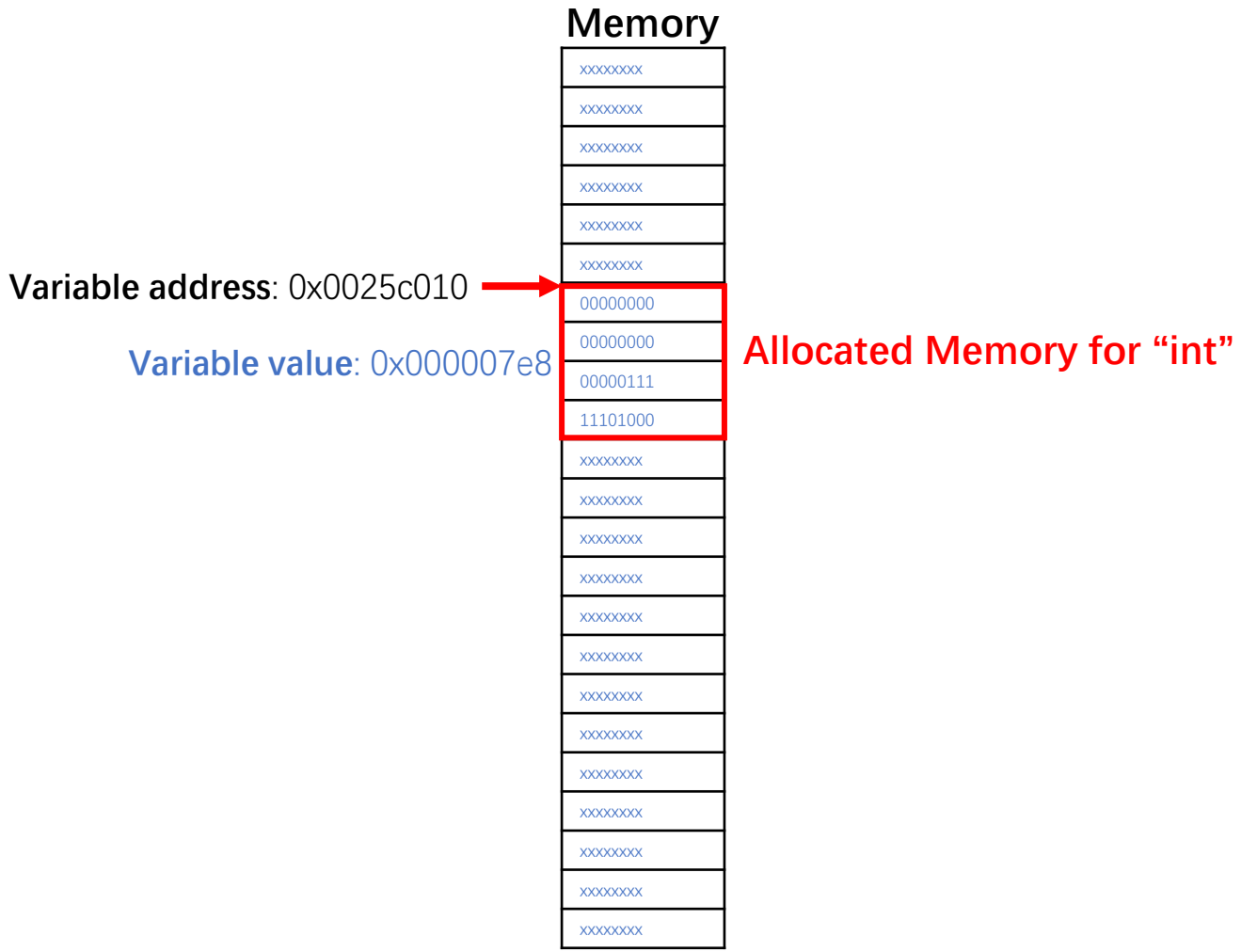
main.cpp

```
1  #include<iostream>
2
3  int main()
4  {
5      int my_number = 2024;
6
7      char my_text[20] = "Hello world ";
8
9      std::cout << my_text << my_number;
10
11     return 0;
12 }
13
14
```

Ln: 14, Col: 1

Run | Share | Command Line Arguments

```
Hello world 2024

** Process exited - Return Code: 0 **
```



https://www.onlinegdb.com/

Basic data types

- Boolean type **`bool`**
- Integer type (signed and unsigned)

`signed char`	`unsigned char`
`signed short int`	`unsigned short int`
`signed int`	`unsigned int`
`signed long int`	`unsigned long int`
`signed long long int`	`unsigned long long int`

- Floating-point types

```
float
double
long double
```

- Character types **`char`**
  (Note that **char** is distinct type from **signed char** and **unsigned char**)

Basic data types

- Boolean type **`bool`**
- Integer type (signed and unsigned)

`signed char`	`unsigned char`
`signed short int`	`unsigned short int`
`signed int`	`unsigned int`
`signed long int`	`unsigned long int`
`signed long long int`	`unsigned long long int`

"signed" may be omitted for names of signed numerical integer types

- Floating-point types **`float`** **`double`** **`long double`**
- Character types **`char`**
  (Note that **char** is distinct type from **signed char** and **unsigned char**)

Get the size of allocated memory space

- **sizeof** operator determines the size, in bytes, of a variable or data type;
- Can be used to get the size of classes, structures, unions and any other user defined data type.

main.cpp

```cpp
#include <iostream>

int main() {
   std::cout << "Size of bool : " << sizeof(bool) << std::endl;
   std::cout << "Size of char : " << sizeof(char) << std::endl;
   std::cout << "Size of short int : " << sizeof(short int) << std::endl;
   std::cout << "Size of int : " << sizeof(int) << std::endl;
   std::cout << "Size of long int : " << sizeof(long int) << std::endl;
   std::cout << "Size of long long int : " << sizeof(long long int) << std::endl;
   std::cout << "Size of float : " << sizeof(float) << std::endl;
   std::cout << "Size of double : " << sizeof(double) << std::endl;
   std::cout << "Size of long double : " << sizeof(long double) << std::endl;

   return 0;
}
```

Ln: 1, Col: 20

Run | Share | Command Line Arguments

```
Size of bool : 1
Size of char : 1
Size of short int : 2
Size of int : 4
Size of long int : 8
Size of long long int : 8
Size of float : 4
Size of double : 8
Size of long double : 16
```

Basic data types

- Boolean type **`bool`** 1 Bytes

- Integer type (signed and unsigned)

		Typical size (differs by systems)
`signed char`	`unsigned char`	1 Bytes
`signed short int`	`unsigned short int`	2 Bytes
`signed int`	`unsigned int`	4 Bytes
`signed long int`	`unsigned long int`	8 Bytes
`signed long long int`	`unsigned long long int`	8 Bytes

- Floating-point types

`float`	4 Bytes
`double`	8 Bytes
`long double`	16 Bytes

- Character types **`char`** 1 Bytes
  (Note that **char** is distinct type from **signed char** and **unsigned char**)

Character encoding

ASCII Chart

Character encoded in 1 Byte

ASCII stands for American Standard Code for Information Interchange, and it defines a particular way to represent English characters (plus a few other symbols) as numbers between 0 and 127

Code	Symbol	Code	Symbol	Code	Symbol	Code	Symbol
0	NUL (null)	32	(space)	64	@	96	`
1	SOH (start of header)	33	!	65	A	97	a
2	STX (start of text)	34	”	66	B	98	b
3	ETX (end of text)	35	#	67	C	99	c
4	EOT (end of transmission)	36	$	68	D	100	d
5	ENQ (enquiry)	37	%	69	E	101	e
6	ACK (acknowledge)	38	&	70	F	102	f
7	BEL (bell)	39	’	71	G	103	g
8	BS (backspace)	40	(	72	H	104	h
9	HT (horizontal tab)	41	)	73	I	105	i
10	LF (line feed/new line)	42	*	74	J	106	j
11	VT (vertical tab)	43	+	75	K	107	k
12	FF (form feed / new page)	44	,	76	L	108	l
13	CR (carriage return)	45	-	77	M	109	m
14	SO (shift out)	46	.	78	N	110	n
15	SI (shift in)	47	/	79	O	111	o
16	DLE (data link escape)	48	0	80	P	112	p
17	DC1 (data control 1)	49	1	81	Q	113	q
18	DC2 (data control 2)	50	2	82	R	114	r
19	DC3 (data control 3)	51	3	83	S	115	s
20	DC4 (data control 4)	52	4	84	T	116	t
21	NAK (negative acknowledge)	53	5	85	U	117	u
22	SYN (synchronous idle)	54	6	86	V	118	v
23	ETB (end of transmission block)	55	7	87	W	119	w
24	CAN (cancel)	56	8	88	X	120	x
25	EM (end of medium)	57	9	89	Y	121	y
26	SUB (substitute)	58	:	90	Z	122	z
27	ESC (escape)	59	;	91	[	123	{
28	FS (file separator)	60	<	92	\	124	|
29	GS (group separator)	61	=	93	]	125	}
30	RS (record separator)	62	>	94	^	126	~
31	US (unit separator)	63	?	95	_	127	DEL (delete)

ASCII Chart

main.cpp

```cpp
#include <iostream>

int main()
{
    for (char c='A'; c <= '^'; ++c)
        std::cout << c << ' ';
}
```

Ln: 1, Col: 20

Run | Share | Command Line Arguments

```
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 [ \ ] ^
```

Code	Symbol	Code	Symbol	Code	Symbol	Code	Symbol
0	NUL (null)	32	(space)	64	@	96	`
1	SOH (start of header)	33	!	65	A	97	a
2	STX (start of text)	34	"	66	B	98	b
3	ETX (end of text)	35	#	67	C	99	c
4	EOT (end of transmission)	36	$	68	D	100	d
5	ENQ (enquiry)	37	%	69	E	101	e
6	ACK (acknowledge)	38	&	70	F	102	f
7	BEL (bell)	39	'	71	G	103	g
8	BS (backspace)	40	(	72	H	104	h
9	HT (horizontal tab)	41	)	73	I	105	i
10	LF (line feed/new line)	42	*	74	J	106	j
11	VT (vertical tab)	43	+	75	K	107	k
12	FF (form feed / new page)	44	,	76	L	108	l
13	CR (carriage return)	45	-	77	M	109	m
14	SO (shift out)	46	.	78	N	110	n
15	SI (shift in)	47	/	79	O	111	o
16	DLE (data link escape)	48	0	80	P	112	p
17	DC1 (data control 1)	49	1	81	Q	113	q
18	DC2 (data control 2)	50	2	82	R	114	r
19	DC3 (data control 3)	51	3	83	S	115	s
20	DC4 (data control 4)	52	4	84	T	116	t
21	NAK (negative acknowledge)	53	5	85	U	117	u
22	SYN (synchronous idle)	54	6	86	V	118	v
23	ETB (end of transmission block)	55	7	87	W	119	w
24	CAN (cancel)	56	8	88	X	120	x
25	EM (end of medium)	57	9	89	Y	121	y
26	SUB (substitute)	58	:	90	Z	122	z
27	ESC (escape)	59	;	91	[	123	{
28	FS (file separator)	60	<	92	\	124	|
29	GS (group separator)	61	=	93	]	125	}
30	RS (record separator)	62	>	94	^	126	~
31	US (unit separator)	63	?	95	_	127	DEL (delete)

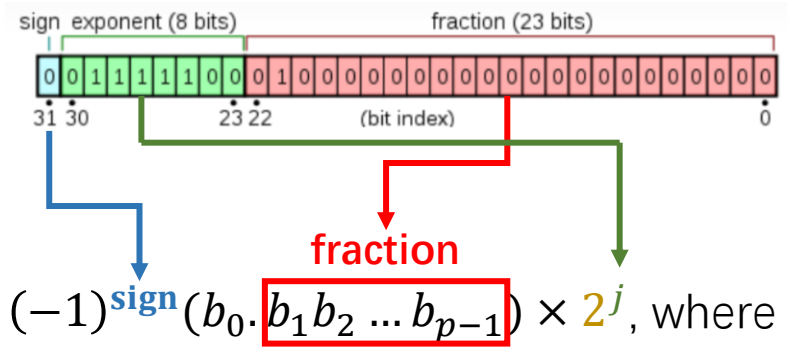
Floating-point types

A floating-point number = s × r ^{j} (Expressions with exponents)
Significand s with ***p*** significant bits
Exponent j
Radix r, and typically **r = 2**

For a fixed **radix r**, representation of a particular number is not unique if no constraints placed on **s** and **j**; to maximize the number of significant digits in significand, **s** and **j** are usually chosen such that first nonzero digit in significand is to **immediate left of radix point**: $1 \le |s| < r$ (typically $1 \le |s| < 2$)

Float: 4 Bytes (32 bits)

How to store a float number:



Example:

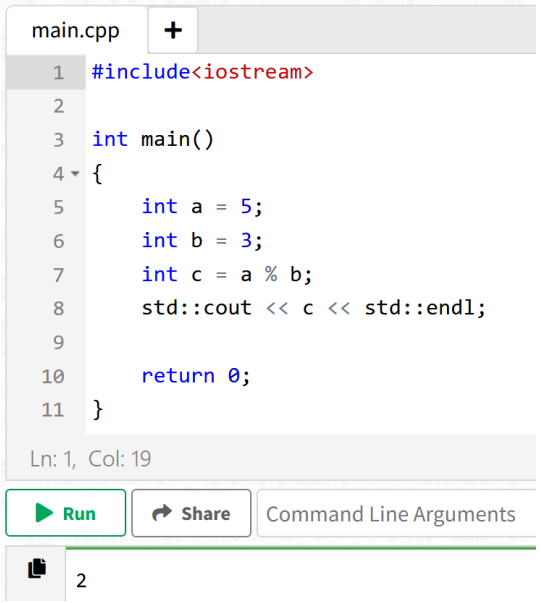
$$(-1)^{\text{sign}}(b_0.b_1b_2 \ldots b_{p-1}) \times 2^{j}, \text{ where sign} \in \{0,1\}, b_i \in \{0,1\}$$

$$0.75 = 0.5 + 0.25 = 0.11_b = (-1)^0\ 1.1_b \times 2^{-1}$$
$$1.25 = 1 + 0.25 = 1.01_b = (-1)^0\ 1.01_b \times 2^0$$
$$-0.5 = -0.1_b = (-1)^1\ 1.0_b \times 2^{-1}$$

Data types: Allowed operations

- Integer type: **+ - * / %**

  addition, subtraction, multiplication, division, modulo

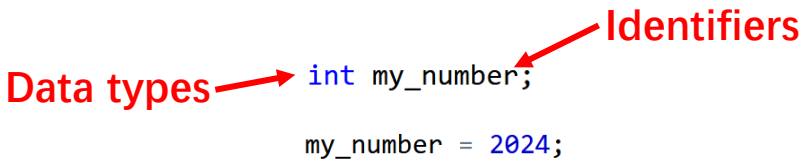


- Floating-point type: **+ - * /**

- Character type: like integers (ASCII)

Variable definition

- Definition introduces identifier for variable




```
int my_number = 2024;
```

- Each identifier must be declared before it can be used
- Variables of the same type can be declared together

```
int a, b, c;
```

Assign values to variables

- Initialize a variable

```
int my_number;

my_number = 2024;
```

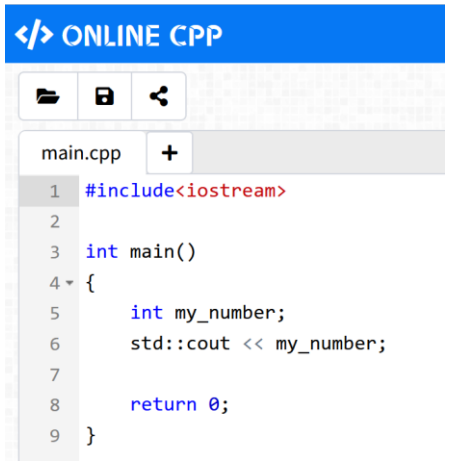
Literals

Use “=” for Assignment

```
int my_number = 2024;
```

Literals

- What is value stored inside a variable if it is declared but not initialized?



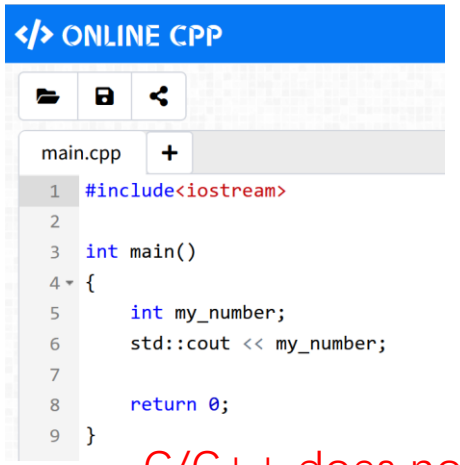
Assign values to variables

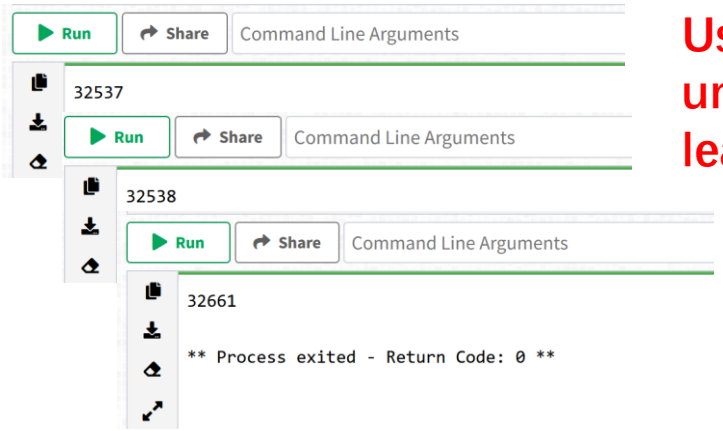
- Initialize a variable

```
int my_number;

my_number = 2024;
```

```
int my_number = 2024;
```

- What is value stored inside a variable if it is declared but **not initialized**?





Using the values of uninitialized variables can lead to unexpected results

C/C++ does not automatically initialize most variables to a given value, the default value is whatever (garbage) value happens to already be in that memory address!

Assign values to variables

- Simple assignment

```
int a;
a = 100;
```

Never forget the semicolon!

- Multiple assignment

```
int a, b, c;
a = b = c = 100;
a = (b = (c = 100));
```

- Compound assignment (more details in Chapter 3)

```
int a = 100;
a += 5;
```

Assign values to variables

- Consistency of data types

main.cpp

```cpp
#include<iostream>

int main()
{
    double a;
    a = 1.414f;
    std::cout << a;

    return 0;
}
```

Ln: 1, Col: 1 (19 selected)

Run | Share | Command Line Arguments

```
1.414

** Process exited - Return Code: 0 **
```

Inconsistent data type!

main.cpp

```cpp
#include<iostream>

int main()
{
    int a;
    a = 1.414f;
    std::cout << a;

    return 0;
}
```

Ln: 1, Col: 19

Run | Share | Command Line Argume

```
1

** Process exited - Return Code: 0 *
```

main.cpp

```cpp
#include<iostream>

int main()
{
    int a;
    a = "Hello world";
    std::cout << a;

    return 0;
}
```

Ln: 1, Col: 19

Run | Share | Command Line Arguments

```
main.cpp: In function 'int main()':
main.cpp:6:9: error: invalid conversion from 'const char*' to 'int' [-fpermissive]
    6 |     a = "Hello world";
      |         ^~~~~~~~~~~~~
      |         |
      |         const char*


** Process exited - Return Code: 1 **
```

HOMEWORK

Homework 2

- 1.Which of the following are valid C++ variable names? If valid, do you think the name is good mnemonic(i.e. reminds you of its purpose)?

  (a) stock_code (b) money$ (c) Jan_Sales (d) X-RAY

  (e) int (f) xyz (g) la (h) invoce_total (i)John's_exam_mark

- 2.Which of the following are valid variable definitions?

  (a) integer account_code; (b) float balance; (c)decimal total;

  (d) int age; (e) double int;

Homework 2

- 3.Write variable definitions for each of the following:

  (a) integer variables number_of_transactions and age_in_years

  (b) floating-point variables total_pay, tax_payment, distance and average

  (c) long integer variables record_position and count

  (d) a character variable account_type

  (e) a double variable gross_pay

Homework 2

- 4. Write a C++ program to assign values to the variables in exercise 3 and display the value of each variable on a separate line.

ADDITIONAL READING

Data type conversion

- Automatic conversion

```
int a = 100;
double b = 100.05;
```

`b = a;` Conversion from **int** to **double**

`a = b;` Conversion from **double** to **int** **Data Loss!**

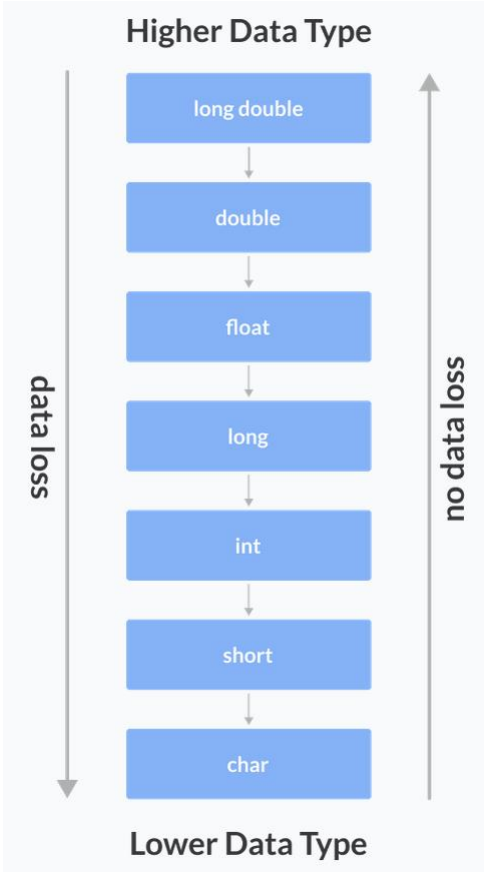
Data type conversion

- Automatic conversion

```
int a = 100;
double b = 100.05;
```

```
b = a;
```
Conversion from **int** to **double**

```
a = b;
```
Conversion from **double** to **int** **Data Loss!**



Data type conversion

- Explicit conversion

```
int a = 100;
double b = 100.05;
```

```
a = (int)b;
```

C-style Type Casting