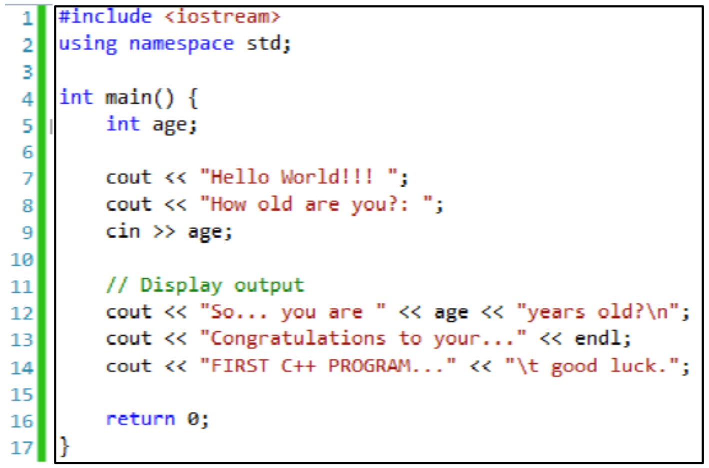
261102
Computer Programming

Lecture 2: Variable & Operator

Variable



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

int main() {
    int age;

    cout << "Hello World!!! ";
    cout << "How old are you?: ";
    cin >> age;

    // Display output
    cout << "So... you are " << age << "years old?\n";
    cout << "Congratulations to your..." << endl;
    cout << "FIRST C++ PROGRAM..." << "\t good luck.";

    return 0;
}
```

Declare ?

Assign ?
Value

Use ?
Stored Value

Note : `using` statements

- Eliminate use of `std::` prefix
- Write `cout` instead of `std::cout`

Variable Declaration

`type variable_name;`

Common data types

`int`	– integer numbers
`double`	– floating point numbers
`bool`	– logical type (true, false)
`char`	– characters

Variable's name

- Series of characters (letters, digits, underscores "_")
- Cannot begin with digit.
- Not include "__" (double underscore).
- C++ is case sensitive i.e., a1 and A1 represent different variables.
- Must NOT be a keyword.

```
alignas, alignof, and, and_eq, asm, auto, bitand, bitor, bool, break, case, catch, char, char16_t,
char32_t, class, compl, const, constexpr, 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, 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
```

Variable Declaration

- Location in memory (RAM) where value can be stored
- Declare variables with name and data type before use

```
int integer1;
int integer2;
int sum;
```

Declarations of variables can be placed almost anywhere in a program, but they must appear before their corresponding variables are used in the program.

- Can declare several variables of same type in one declaration by using Comma-separated list

```
int integer1, integer2, sum;
```

Variable's Name

– first_name	valid
– last-name	invalid – contains '-'
– include	invalid – match a keyword
– YearOne	valid
– 40days	invalid – starts with numeric character
– example#1	invalid – contains '#'
– _long	valid
– goto	invalid – match a keyword
– float__a	invalid – contains double underscores
– using	invalid – match a keyword

Value Assignment

- **Input stream object**
  - `std::cin`
  - `>>` (stream extraction operator)
    - Waits for user to input value, then press *Enter* (Return) key
    - Stores value in variable to right of operator
    - Converts value to data type
  - To receive the input value for single variable:

    `std::cin >> variable_name;`

    - variable_name – name of declared variable.
  - To receive the input values for multiple variables:

    `std::cin >> var1 >> var2 >> var3 >> var4;`

    - Each value from input stream must be separated by whitespace (Spacebar, Tab, Enter)
    - Store each value to each variable in left-to-right order

Value Assignment

- **Input stream object**

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

int main()
{
    int a,b,c;
    cout << "Input the 1st number: ";
    cin >> a;
    cout << "Input the 2nd number: ";
    cin >> b;
    cout << "Input the 3rd number: ";
    cin >> c;

    cout << "Average = " << (a+b+c)/3.0 ;

    return 0;
}
```

```
Input the 1st number: 15
Input the 2nd number: 20
Input the 3rd number: 30
Average = 21.6667
```

Value Assignment

- **Input stream object**

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

int main()
{
    int a,b,c;
    cout << "Input 3 numbers: ";
    cin >> a >> b >> c;

    cout << "Average = " << (a+b+c)/3.0 ;

    return 0;
}
```

```
Input 3 numbers: 15 20  30
Average = 21.6667
```

a	b	c
15	20	30

Value Assignment

- **Input stream object**

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

int main()
{
    int a,b,c;
    cout << "Input 3 numbers: ";
    cin >> a;
    cin >> b;
    cin >> c;

    cout << "Average = " << (a+b+c)/3.0 ;

    return 0;
}

```

```
Input 3 numbers: 15 20  30
Average = 21.6667
```

a	b	c
15	20	30

Value Assignment

- **Input stream object**

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

int main()
{
    int a,b,c;
    cout << "Input 3 numbers: ";
    cin >> a >> b >> c;

    cout << "Average = " << (a+b+c)/3.0 ;

    return 0;
}
```

```
Input 3 numbers: 15 20


30
Average = 21.6667
```

Value Assignment

- **Input stream object**

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

int main()
{
    int a,b,c;
    cout << "Input 3 numbers: ";
    cin >> a >> b >> c;

    cout << "Average = " << (a+b+c)/3.0 ;

    return 0;
}
```

```
Input 3 numbers: 15 20 30 40 50 60 70
Average = 21.6667
```

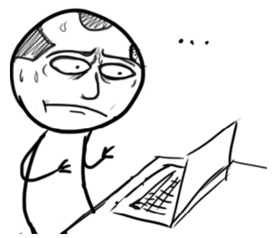
Value Assignment

- **Input stream object**

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

int main()
{
    int a,b,c,x;
    cout << "Input 3 numbers: ";
    cin >> a >> b >> c;

    cout << "Average = " << (a+b+c)/3.0;

    cout << "\nInput another number: ";
    cin >> x;
    cout << "Another number is " << x;

    return 0;
}
```

```
Input 3 numbers: 15 20 30 40 50 60 70
Average = 21.6667
Input another number: Another number is 40
```

อ้าว..ไม่ได้พิมพ์ ข้ามไปเลย



Value Assignment

- **= (Assignment operator)**
  - Assigns value to variable
  - Copy value on the right side to the variable on the left side
  - Example:

```
PI = 3.1416;
sum = variable1 + variable2;
a = b = 5;

int x = a + 1;
double x = 5.5;
char char5 = '5', at = '@';
string mylove = "I love you.";
```

Initialization

Value Assignment

- **= (Assignment operator)**

```
int a;
int b;
a = 1;
b = 1;
```
✓

```
int a,b;
a = 1;
b = 1;
```
✓

```
int a, b;
a = b = 1;
```
✓

```
int a = 1, b;
b = 1;
```
✓

```
int a, b = 1;
a = 1;
```
✓

```
int a = 1, b = 1;
```
✓

```
int a = b = 1;
```
✗

```
int a = int b = 1;
```
✗

```
int b;
int a = b = 1;
```
✓

Fundamental Data 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**	

*only the part not in italics is required to identify the type

http://www.cplusplus.com/doc/tutorial/variables/

Fundamental Data Types

Character types

- Data type that hold exactly one character
- Character literal must be enclosed within ' ' (single quote)

  'A', 'a', '5', '.', '_', ' ' *(space)*

  '\t' *(tab)*, '\n' *(newline)*

  Integer value represented as character in single quotes **'z'** is integer value of alphabet **z** (**122** in ASCII)

- Character variable declaration, assignment and usage

  `char c;` ←Declare variable named c as character type

  `char mygrade = 'F';` ←Declare variable named mygrade as character type and assign it value to alphabet F

  `mygrade == 'A';` ←Compare value of variable mygrade whether it is alphabet A or not? (Use equality operator ==)

Fundamental Data Types

Character types

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

int main() {
    char A = '+';
    char x = A;
    char y = 'A';

    cout << "x = " << x << endl;
    cout << "y = " << y << endl;

    return 0;
}
```

```
x = +
y = A
```

Fundamental Data Types

Numerical Integer types

- Used for store an integer.
- Integer literal (fixed value in source code):

Decimal	17, 1024, 65535, -127, 0 17u (unsigned int) 17l (long) 17ul (unsigned long)
Binary	0b10001, 0b1111111, 0b000011
Octal	021, 0113, 0720
Hexadecimal	0x11, 0x4b, 0xFF

http://en.cppreference.com/w/cpp/language/integer_literal

Fundamental Data Types

Numerical Integer types

- Examples of integer variable declaration, assignment and usage

```cpp
int x, y, z = 0;
unsigned short pix = 0xFF;
x = z + 10 – 0b10;
x == -55;
```

Fundamental Data Types

Numerical Integer types

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

int main() {
    int a = 100, b = 0100, c = 0b100, d = 0x100;

    cout << "a = " << a << endl;
    cout << "b = " << b << endl;
    cout << "c = " << c << endl;
    cout << "d = " << d << endl;

    return 0;
}
```

```
a = 100
b = 64
c = 4
d = 256
```

Fundamental Data Types

Numerical Integer types

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

int main() {
    int a = 10, b = 10.5;
    unsigned short c = 0xFFFF, d = 0xFFFFFFFF;

    cout << "a = " << a << endl;
    cout << "b = " << b << endl;
    cout << "c = " << c << endl;
    cout << "d = " << d << endl;

    return 0;
}
```

```
a = 10
b = 10
c = 65535
d = 65535
```

Fundamental Data Types

Floating-point types

- Representing numbers that have fractional part.
- Floating-point literal $\longrightarrow$
- Floating-point variable declaration, assignment and usage

Default type - double 3.14159, 0.01 6.02e23, 1.75e-9
3.14159L (long double)
6.02e23f (float)

```
double x, y = 12.345;
const float c = 3e8;
x = c*y;
x >= 9.9e9;
cout << x;
```

Fundamental Data Types

Boolean type

- Representing logical data (true, false).
  - false (0)
  - true (any values other than 0)
- Example of variable declaration, assignment and usage

```
bool isKak;
isKak = score < 55;
bool narak = true;
isKak&&(!narak)
```

Memory Concepts

– Variable names correspond to actual locations in computer's memory

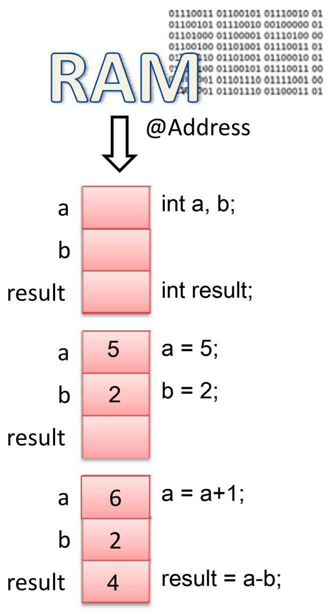
– Every variable has name, type, size and value

– When new value placed into variable, overwrites previous value

– Reading variables from memory nondestructive

Memory Concepts

```
// operating with variables

#include <iostream>
using namespace std;

int main ()
{
  // declaring variables:
  int a, b;
  int result;

  // process:
  a = 5;
  b = 2;
  a = a + 1;
  result = a - b;

  // print out the result:
  cout << result;

  // terminate the program:
  return 0;
}
```



Type Casting

(type_name) expression — C-like cast notation

type_name (expression) — Functional cast notation

```
#include <iostream>

int main() {

   int sum = 17, count = 5;
   double mean;

   mean = sum / count;
   std::cout << "Value of mean : " << mean ;

   return 0;
}
```

```
Value of mean : 3
```

```
#include <iostream>

int main() {

   int sum = 17, count = 5;
   double mean;

   mean = (double) sum / count;
   std::cout << "Value of mean : " << mean ;

   return 0;
}
```

```
Value of mean : 3.4
```

Constant Expression

const type_name variable_name = value;

– Use const type qualifier **const** to defines that the data is constant (is not modifiable).

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

int main() {
    const float PI = 3.1416;
    const char nl = '\n';

    cout << setw(16) << "r = 1: Area = " << PI*1*1 << nl;
    cout << setw(16) << "r = 1.5: Area = " << PI*1.5*1.5 << nl;
    cout << setw(16) << "r = 2: Area = " << PI*2*2 << nl;
    cout << setw(16) << "r = 2.5: Area = " << PI*2.5*2.5 << nl;

    return 0;
}
```

```
  r = 1: Area = 3.1416
r = 1.5: Area = 7.0686
  r = 2: Area = 12.5664
r = 2.5: Area = 19.635
```

Constant Expression

```
6       const float PI;
```

```
 In function 'int main()':
6:17: error: uninitialized const 'PI' [-fpermissive]
```

```
6       const float PI = 3.1416;
7       const char nl = '\n';
8
9       PI = 3.14159265358979323846264338327950;
```

```
 In function 'int main()':
9:8: error: assignment of read-only variable 'PI'
```

Operators

Operators are used to process variables and literals

- Assignment Operator (=)
- Arithmetic Operators (+ - * / %)
- Compound Assignment Operators (+= -= *= /= %=)
- Increment & Decrement Operators (++ --)
- Relational & Equality Operators (< > <= >= == !=)
- Logical Operators (! && ||)
- Bitwise Operators (& | ^ ~ << >>)
- Pointer Operators (& *)
- ….

Arithmetic & Compound Assignment Operators

- **Arithmetic Operators**

  (+, -, *, /, %)

operator	description
+	addition
-	subtraction
*	multiplication
/	division
%	modulo

- **Compound Assignment**

  (+=, -=, *=, /=, %=, >>=, <<=, &=, ^=, |=)

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);`

Increment & Decrement Operators

– Increment operator (++) increases the value stored in a variable by one (equivalent to +=1)

– Decrement operator (--) decreases the value stored in a variable by one (equivalent to -=1)

– ++x; and x+=1; and x=x+1; are equivalent expressions

– Can be used both as a prefix (++x) and as a suffix (x++)

- prefix (++x) = the expression evaluates to the final value of x (already increased)
- suffix (x++) = the value is also increased, but the expression evaluates to the value that x had before being increased.

Increment & Decrement Operators

```
int x = 3;
cout << x <<'\n';
cout << ++x <<'\n';
cout << x;
```

```
3
4
4
```

(x increases before cout)

```
int x = 3;
cout << x <<'\n';
cout << x++ <<'\n';
cout << x;
```

```
3
3
4
```

(x increases after cout)

```
int y,x = 3;
y = ++x;
cout << "x = " << x << ", y = " << y;
```

(x increases before assign value to y)

```
x = 4, y = 4
```

```
int y,x = 3;
y = x++;
cout << "x = " << x << ", y = " << y;
```

(x increases after assign original value to y)

```
x = 4, y = 3
```

Increment & Decrement Operators

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

int main()
{
    int x = 3;
    cout << 2+(++x)++ << '\n';
    cout << x ;
    return 0;
}
```

?

Relational & Equality Operators

– The result of rational and equality operation is either true or false (i.e., a Boolean value)

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

```
(7 == 5)     // evaluates to false
(5 > 4)      // evaluates to true
(3 != 2)     // evaluates to true
(6 >= 6)     // evaluates to true
(5 < 5)      // evaluates to false
```

Logical Operators

– The result of rational and equality operation is either true or false
  - Value of 0 is consider as false
  - Any values other than 0 is considered as true

– The operator ! is the Boolean operation NOT

– The operator && corresponds to the operation AND

– The operator || corresponds to the operation OR

&& OPERATOR (and)		
a	b	a && b
true	true	true
true	false	false
false	true	false
false	false	false

|| OPERATOR (or)		
a	b	a || b
true	true	true
true	false	true
false	true	true
false	false	false

Logical Operators

```
!(5 == 5)   // evaluates to false because the expression at its right (5 == 5) is true
!(6 <= 4)   // evaluates to true because (6 <= 4) would be false
!true       // evaluates to false
!false      // evaluates to true
```

```
( (5 == 5) && (3 > 6) )  // evaluates to false ( true && false )
( (5 == 5) || (3 > 6) )  // evaluates to true ( true || false )
```

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).

true ignored
((5 == 5) || (3 > 6))

true

false ignored
((5 != 5) && (3 > 6))

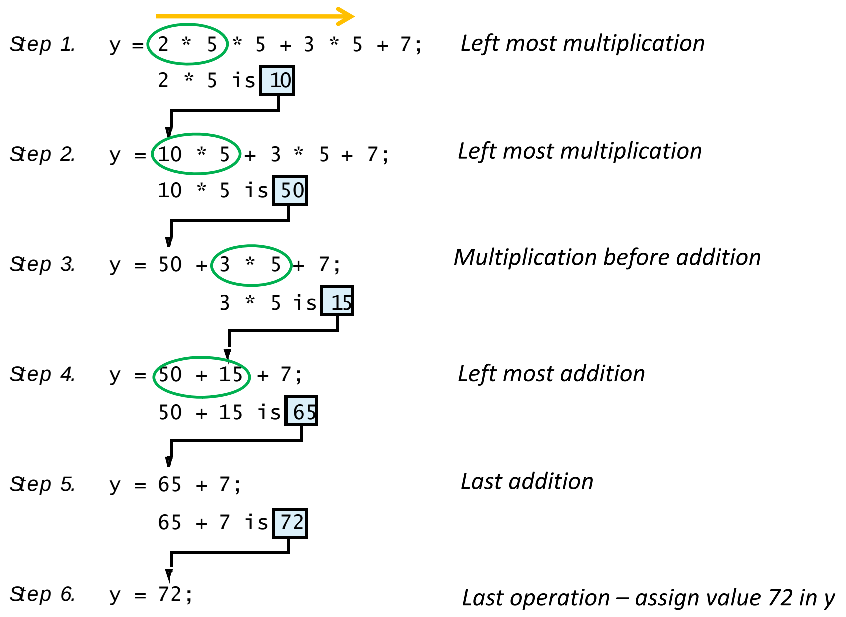
false

Precedence of Operators

Precedence	Operator	Description	Associativity
1	()	Parentheses	Left-to-right
2	++ --	Suffix/postfix increment and decrement	
3	++ --	Prefix increment and decrement	Right-to-left
	!	Logical NOT	
	(*type*)	C-style cast	
4	* / %	Multiplication, division, and remainder	Left-to-right
5	+ -	Addition and subtraction	
6	< <=> >=	Relational operators	
7	== !=	Relational operators	
8	&&	Logical AND	
9	||	Logical OR	
10	=	Direct assignment (provided by default for C++ classes)	Right-to-left
	+= -=	Compound assignment by sum and difference	
	*= /= %=	Compound assignment by product, quotient, and remainder	

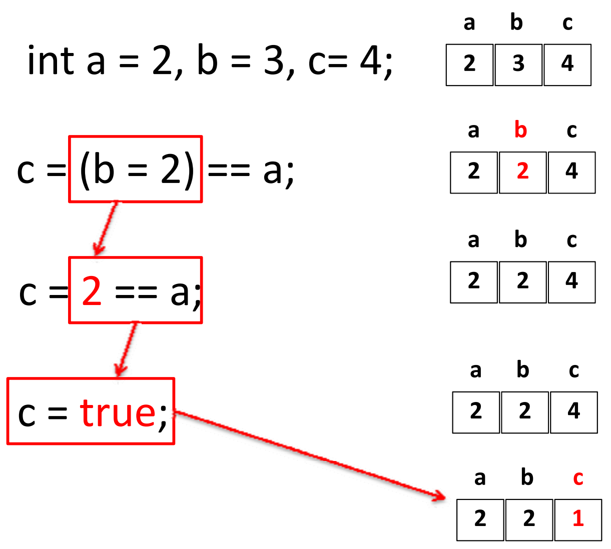
http://en.cppreference.com/w/cpp/language/operator_precedence

Precedence of Operators



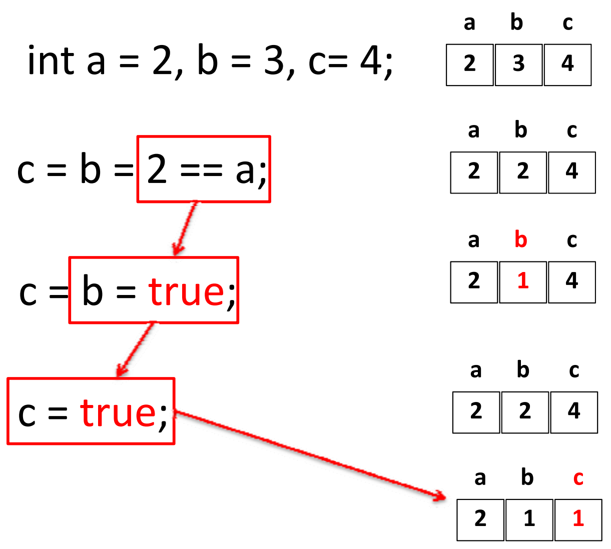
```
Step 1.   y = 2 * 5 * 5 + 3 * 5 + 7;      Left most multiplication
              2 * 5 is 10
Step 2.   y = 10 * 5 + 3 * 5 + 7;         Left most multiplication
              10 * 5 is 50
Step 3.   y = 50 + 3 * 5 + 7;             Multiplication before addition
                   3 * 5 is 15
Step 4.   y = 50 + 15 + 7;                Left most addition
              50 + 15 is 65
Step 5.   y = 65 + 7;                     Last addition
              65 + 7 is 72
Step 6.   y = 72;                         Last operation – assign value 72 in y
```

Precedence of Operators



```
int a = 2, b = 3, c= 4;
c = (b = 2) == a;
c = 2 == a;
c = true;
```

Precedence of Operators



```
int a = 2, b = 3, c= 4;
c = b = 2 == a;
c = b = true;
c = true;
```

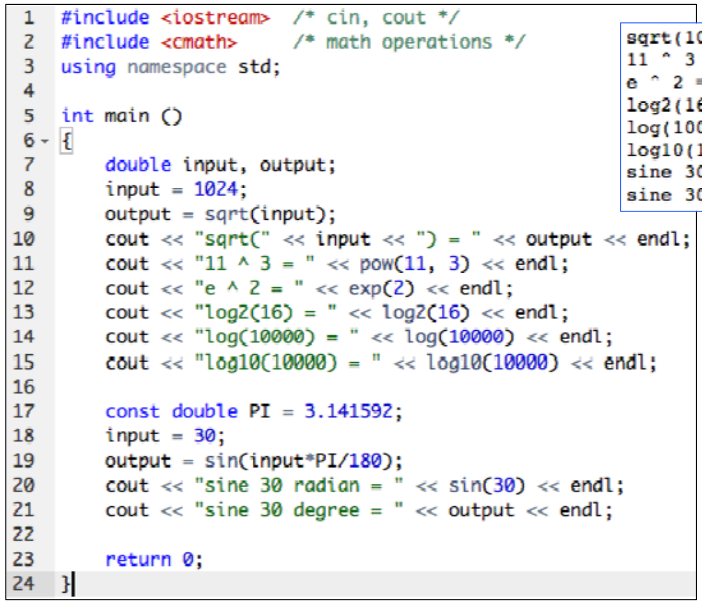
Mathematical Functions with C++

- Standard library header <cmath> declares a set of functions to compute common mathematical operations

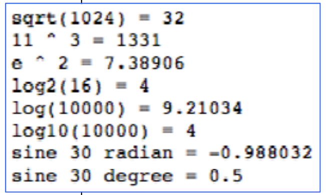
```
#include <cmath>
```

- Some examples of mathematical functions in <cmath>
  - Trigonometric functions – sin, cos, tan, asin, acos, …
  - Exponential and Logarithmic functions – exp, log, log2, log10, …
  - Power functions – pow, sqrt, …
  - Rounding functions – ceil, floor
  - Other functions – abs, …

Mathematical Functions with C++



```cpp
#include <iostream>   /* cin, cout */
#include <cmath>      /* math operations */
using namespace std;

int main ()
{
    double input, output;
    input = 1024;
    output = sqrt(input);
    cout << "sqrt(" << input << ") = " << output << endl;
    cout << "11 ^ 3 = " << pow(11, 3) << endl;
    cout << "e ^ 2 = " << exp(2) << endl;
    cout << "log2(16) = " << log2(16) << endl;
    cout << "log(10000) = " << log(10000) << endl;
    cout << "log10(10000) = " << log10(10000) << endl;

    const double PI = 3.141592;
    input = 30;
    output = sin(input*PI/180);
    cout << "sine 30 radian = " << sin(30) << endl;
    cout << "sine 30 degree = " << output << endl;

    return 0;
}
```



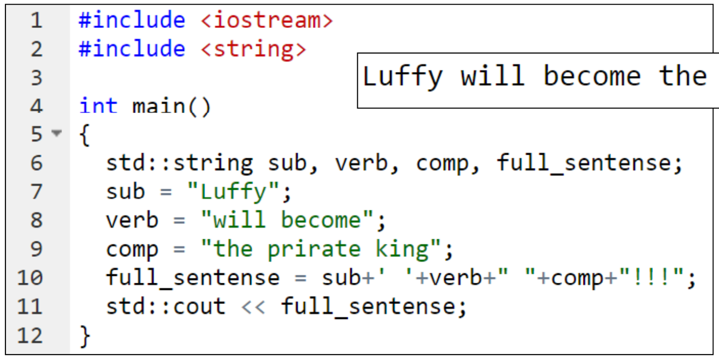
```
sqrt(1024) = 32
11 ^ 3 = 1331
e ^ 2 = 7.38906
log2(16) = 4
log(10000) = 9.21034
log10(10000) = 4
sine 30 radian = -0.988032
sine 30 degree = 0.5
```

String (std::string)

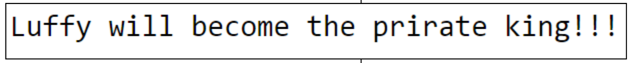
- Series of characters treated as single unit
- Can include letters, digits, special characters **+**, **-**, ***** …
- String literal enclosed in **" "** (double quotes) , for example:

  "I like C++"

- #include <string> in preprocessor

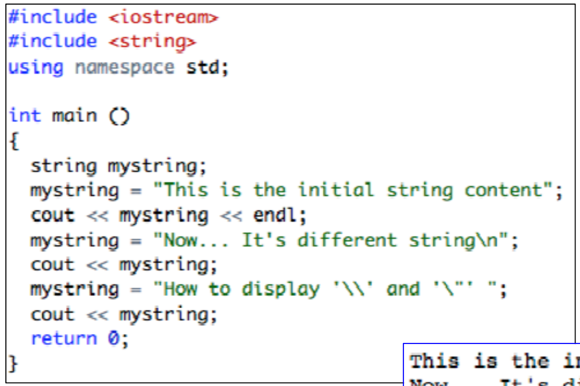


```cpp
#include <iostream>
#include <string>

int main()
{
  std::string sub, verb, comp, full_sentense;
  sub = "Luffy";
  verb = "will become";
  comp = "the prirate king";
  full_sentense = sub+' '+verb+" "+comp+"!!!";
  std::cout << full_sentense;
}
```

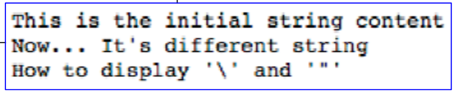


```
Luffy will become the prirate king!!!
```

String (std::string)



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

int main ()
{
  string mystring;
  mystring = "This is the initial string content";
  cout << mystring << endl;
  mystring = "Now... It's different string\n";
  cout << mystring;
  mystring = "How to display '\\' and '\"' ";
  cout << mystring;
  return 0;
}
```



```
This is the initial string content
Now... It's different string
How to display '\' and '"'
```

Example 2-A: Chat with Fahsai

สวัสดีค่ะ เราชื่อ ฟ้าใส

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

int main()
{
    int year;
    cout << "Fahsai: Sawadee ka...\n";
    cout << "Fahsai: I want to know what year were you born in?\n";
    cout << "Me: ";
    cin >> year;
    cout << "Fahsai: Wow!!! You're so young.\n";
    cout << "Fahsai: You're just only " << 2016-year << " years old.\n" ;

    string ans;
    cout << "Fahsai: Do you like to watch movies?\n";
    cout << "Me: ";
    cin >> ans;
    cout << "Fahsai: What is your best movie in the last year?\n";
    cout << "Me: ";
    cin >> ans;
    cout << "Fahsai: I think so. " << ans << " was a really good movie!!!" ;

    return 0;
}
```

Example 2-A: Chat with Fahsai

```
Fahsai: Sawadee ka...
Fahsai: I want to know what year were you born in?
Me: 1998
Fahsai: Wow!!! You're so young.
Fahsai: You're just only 18 years old.
Fahsai: Do you like to watch movies?
Me: yes
Fahsai: What is your best movie in the last year?
Me: Maebia
Fahsai: I think so. Maebia was a really good movie!!!
```

Example 2-A: Chat with Fahsai

```
Fahsai: Sawadee ka...
Fahsai: I want to know what year were you born in?
Me: 1987
Fahsai: Wow!!! You're so young.
Fahsai: You're just only 29 years old.
Fahsai: Do you like to watch movies?
Me: yes
Fahsai: What is your best movie in the last year?
Me: Star Wars
Fahsai: I think so. Star was a really good movie!!!
```

Example 2-A: Chat with Fahsai

`getline(cin, string_variable);`

`cin.ignore();`

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

int main()
{
    int year;
    cout << "Fahsai: Sawadee ka...\n";
    cout << "Fahsai: I want to know what year were you born in?\n";
    cout << "Me: ";
    cin >> year;
    cin.ignore();
    cout << "Fahsai: Wow!!! You're so young.\n";
    cout << "Fahsai: You're just only " << 2016-year << " years old.\n" ;

    string ans;
    cout << "Fahsai: Do you like to watch movies?\n";
    cout << "Me: ";
    getline(cin, ans);
    cout << "Fahsai: What is your best movie in the last year?\n";
    cout << "Me: ";
    getline(cin, ans);
    cout << "Fahsai: I think so. " << ans << " was a really good movie!!!" ;

    return 0;
}
```

cin >> leaves the '\n' character in the input stream
When switch between >> and getline() use cin.ignore() to discard '\n'

Extracts characters (including space and tab) from the stream until '\n' found

Example 2-A: Chat with Fahsai

```
Fahsai: Sawadee ka...
Fahsai: I want to know what year were you born in?
Me: 1987
Fahsai: Wow!!! You're so young.
Fahsai: You're just only 29 years old.
Fahsai: Do you like to watch movies?
Me: not much
Fahsai: What is your best movie in the last year?
Me: Star wars
Fahsai: I think so. Star wars was a really good movie!!!
```