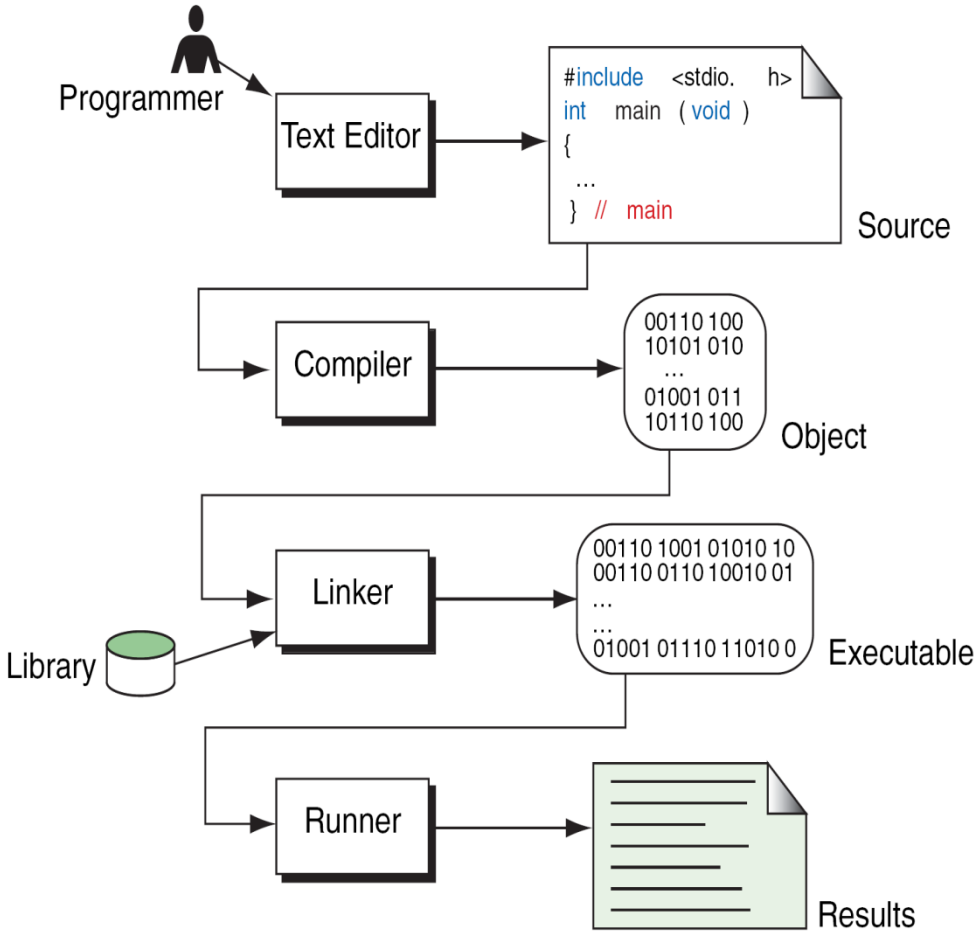
CS2313 Computer Programming

LT2 – Language Syntax, Variable, Data types and Basic I/O-Part I

Building a C++ Program

- **Writing** source code in C++.
  - e.g. hello.cpp
- **Preprocessing**
  - Processes the source code for compilation.
- **Compilation**
  - Checks the grammatical rules (syntax).
  - Source code is converted to object code in machine. language (e.g. hello.obj).
- **Linking**
  - Combines object code and libraries to create an executable (e.g. hello.exe).
  - Library: common functions (input, output, math, etc).



Simple Program

```
/* The traditional first program in honor of
   Dennis Ritchie who invented C at Bell Labs
   in 1972 */

#include <iostream>
using namespace std;
void main()
{
   cout << "Hello, world!\n";
}
```

Includes a file

Specifies namespace

main function, where to start

Starts a new line

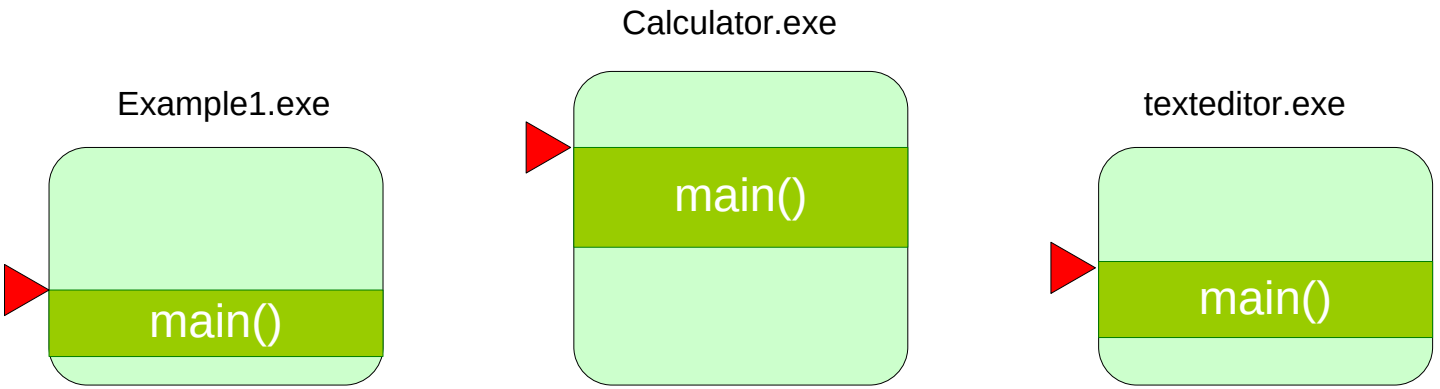
Double quotation marks

Output stream object
Stream insertion operator

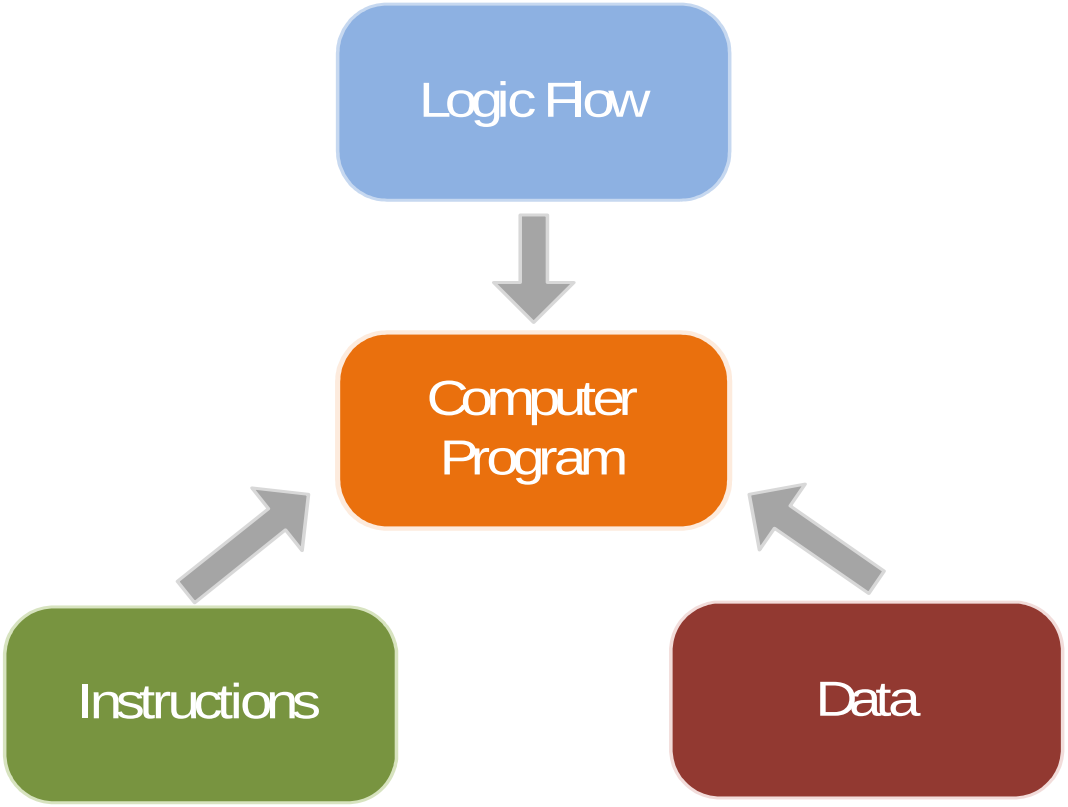
Function - main

```
int main()
{
    return 0;
}
```

- The starting point of program (the first function called by the computer).

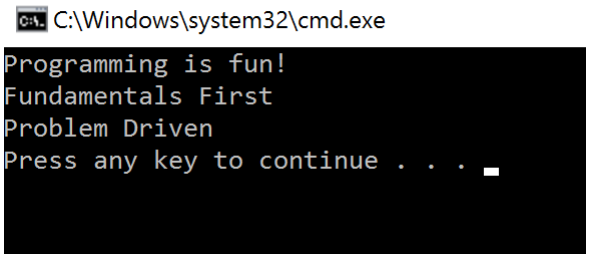


Computer Program



Computer Program

```
#include <iostream>
using namespace std;
int main()
{
  cout << "Programming is fun!" << endl;
  cout << "Fundamentals First" << endl;
  cout << "Problem Driven" << endl;
  return 0;
}
```



stream manipulator

Outlines

- C++ Language Syntax
- Variable type, scope and declaration
- Constant
- Operators
- Basic I/O operation with `cin` and `cout` objects

Outcomes

- Describe the basic syntax and data types of C++ language.
- Explain the concepts of constant, variable and their scope.
- Declare variable and constant under different scopes.
- Perform update on variables via different operators.
- Able to output variables' values to screen with different formats.
- Able to read value from keyboard and assign to variable.

Syntax of C++

- Like any language, C++ has an **alphabet** and **rules** for putting together words and punctuation to make legal program; that is called ***syntax*** of the language.

- C++ compiler detects any **violation** of the syntactic rules within the program.

- C++ compiler collects the characters of the program into ***tokens***, which form the basic vocabulary of the language.

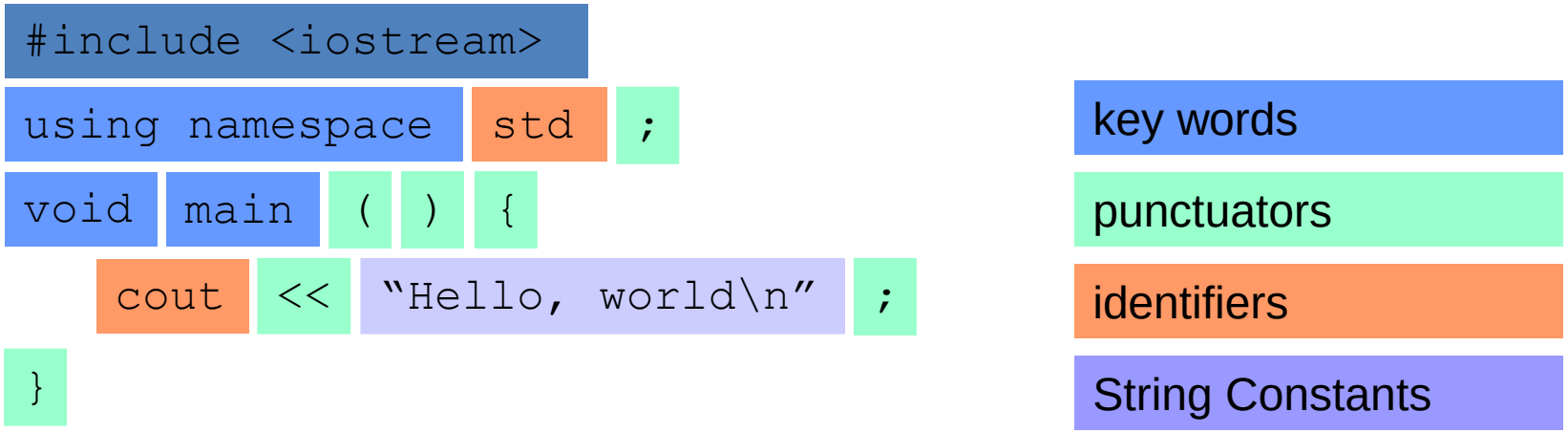
- ***Tokens*** are separated by space.

Syntax - Tokens

- Tokens in C++ can be categorized into:
  - *keywords*, e.g., `main`, `return`, `int`.
  - *identifiers*, e.g., user-defined variables and identifiers used in preprocessing statements.
  - *string constants*, e.g., `"Hello"`.
  - *numeric constants*, e.g., `7`, `11`, `3.14`.
  - *operators*, e.g., `++`.
  - *punctuators*, e.g., `;` and `,`.

Syntax – A Simple Program

```
#include <iostream>
using namespace std;
void main()
{
    cout << "Hello, world!\n";
}
```



Keywords

Keywords

Keywords – reserved words.
the following list those covered in this course:

Data type	`char`	`double`	`float`	`int`	`bool`
	`long`	`short`	`signed`	`unsigned`	`void`
Flow control	`if`	`else`	`switch`	`case`	
	`break`	`default`	`for`	`do`	
	`while`	`continue`			
Others	`using`	`namespace`	`true`	`false`	`sizeof`
	`return`	`const`	`class`	`new`	`delete`
	`operator`	`public`	`protected`	`private`	`friend`
	`this`	`try`	`catch`	`throw`	`struct`
	`typedef`	`enum`	`union`		

Keywords

- Each keyword in C++ has a reserved **meaning** and **cannot** be used as identifiers.

- Most of the keywords given in the previous page will be covered in this course

Identifiers

- Identifiers give ***unique*** names to various objects in a program, like the name of functions and variables.

- Reserved keywords, like `float` and `int`, cannot be used as identifiers.

- An identifier is composed of a sequence of letters, digits and underscore:
  - E.g. `myRecord`, `point3D`, `last_file`.

- An identifier must begin with either an underscore (not recommended) or a letter:
  - **valid** identify: `_income`, `_today_record`, `record1`.
  - **Invalid** identify: `3D_Point`, `2ppl_login`,`-right-`.

Identifiers

- Always use **meaningful** names for identifiers
  - `Data001, data002, data003.`
  - `Localvaribleı1.`

- at least the first 31 characters of an identifier are discriminated.

Variable and Constant

Variable and Constant

- Data stored in memory are in binary format, i.e. 0 / 1.

Variable:

Memory storage whose value will be changed during program execution.

Constant:

Memory storage whose value will **NOT** be changed during program execution.

- Every variable have 3 attributes: *name*, *type* and *scope*

Name:

Identifier of the variable.

Type:

C++ is a strictly typed language, variable/constant must have a data type, either predefined or user-defined.

Scope:

It defines where the variable can be accessed (more detail later).

C++ Predefined Data type

- **Numerical**
  - `int`: Integer type (`1, 3, 8 , 3222, 421`)

```
int x, y;
```

  - `float, double`: real number type (`0.25, 6.45, 3.01e-5`)

```
float x,y;
double z=1.0f;
```

- **Character**
  - `char`: ASCII character (`a, e, o, \n`)

```
char c;
```

- **Logic**
  - `bool`: Boolean (`true,false`)

```
bool b;
```

- **Other**
  - `void`: empty values

```
void main();
```

Variable Declaration

- Variable and constant must be declared before use.
- Format:
  - Data_type identifier (variable name);
- Examples:

```
int age;
float volume;
char initial;
char student_name[];
```

- Optionally, the initial value of a variable can be set with declaration.

```
int age = 18;
float volume = 15.4;
```

Variable Scope – Global vs. Local

Scope of a variable refers to the **accessibility** of a variable.

Global:

- A variable defined in the global declaration sections of a program, i.e., defined outside a function **block**.
- Can be accessed by all functions.

Local:

- Declared in a block and can be only accessed within the block.
- Accessing a local variable outside the block will produce unpredictable result.

Block is zero or more statements enclosed in a set of braces {…block… }.

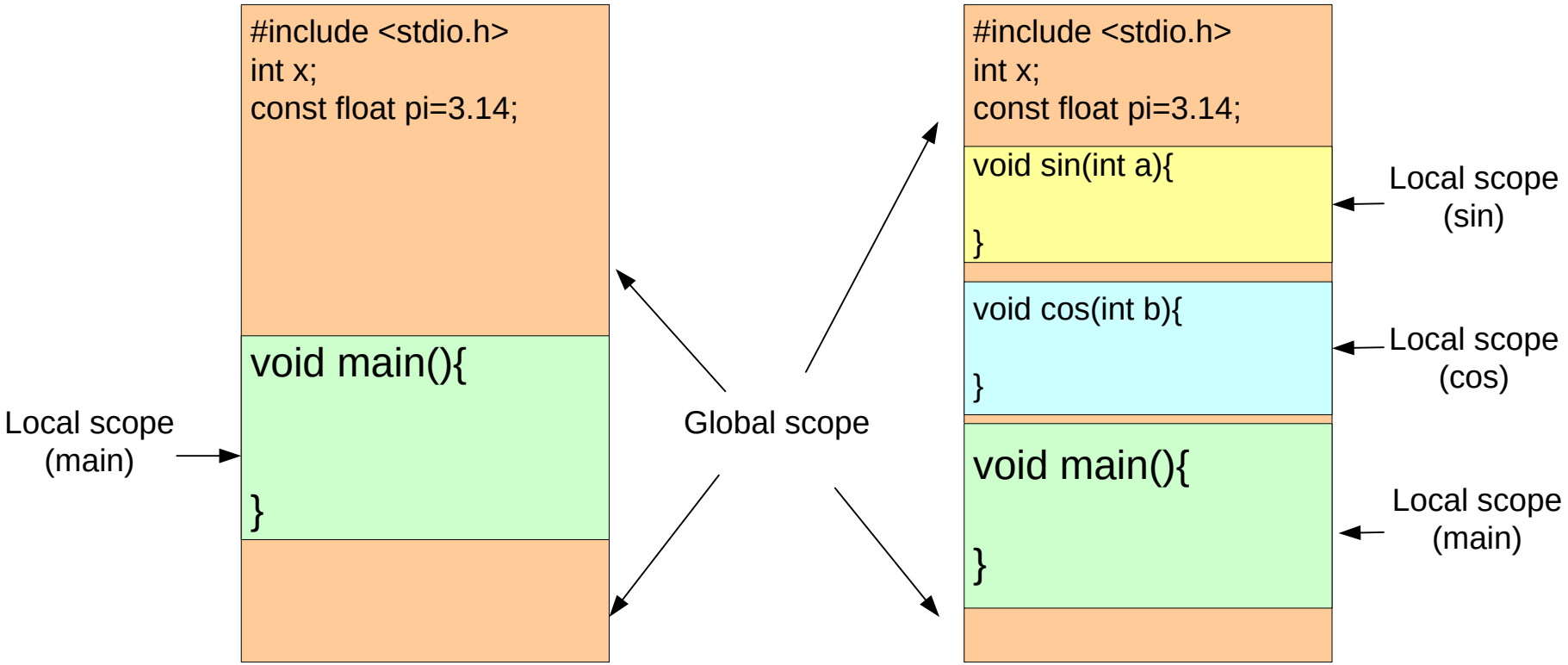
Declaration – Variable (Local)

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

void main(){

  int number1 = 12;
  int number2 = 3;
  int result;

  result=number1+number2;



}
```

Declaration – Variable (Global)

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

int number1 = 12;
int number2 = 3;

void main(){

  int result;

  result=number1+number2;



}
```

Declaration – Global vs. Local



Declaration - Scope

- Scope determines the region of the program in which a defined object is **visible** (can be accessed).
- The concept of **scope** applies to both **variable** and **function**.
- **Block** is zero or more statements enclosed in a set of braces {…block… }.
- A block has a declarations section and a statement section.
- Global area of a program consists of all statements that are **outside** functions.

Declaration - Scope

```
int number1=12;
int number2=3;

int min(int x, int y){
   if (x>y){
    return y;
   }else{
    return x;
   }
}

void main(){
   int result;
   result=min(number1,number2);
}
```

Data Type `int`

- Typically, an `int` is stored in two or four bytes (1 byte = 8 bits).

- The length of an `int` data type restricts the range of number it can store, e.g., a 32-bit `int` can store any number in the range of -2^{31} and $2^{31} - 1$ (with bit 0 used as a sign bit).

- When an `int` is assigned of a value greater than its maximum value, overflow occurs and this ends up with **illogical** result; similarly underflow may occur when value smaller than the minimum value is assigned to a variable of the data type; however, C++ does **NOT** inform you the errors.

Data Type `int`

- Be careful about the range limitations
- Gaming design
  - int money;
  - Condition: money < 2^{31} -1
  - Otherwise, overflow may happen!
  - Solution: double/restrict the range

A better template

```
#include <iostream>
using namespace std;
int main(){
    /* Place your code here! */

  return 0;
}
```

`short`, `long` **and** `unsigned`

- `short`, `long` and `unsigned` are special data types for representing integers.
- `short` is used to conserve space.
- `long` is used to describe large integers (previously the int is 2bytes).
- `unsigned` is of the same size as `int` except it assumes the value to be stored is **positive** and thus the sign bit can be conserved in order to double `int`'s maximum value that unsigned can store, e.g., the range of data a 32-bit unsigned can store any number in the range of 0 and 2^{32} -1.

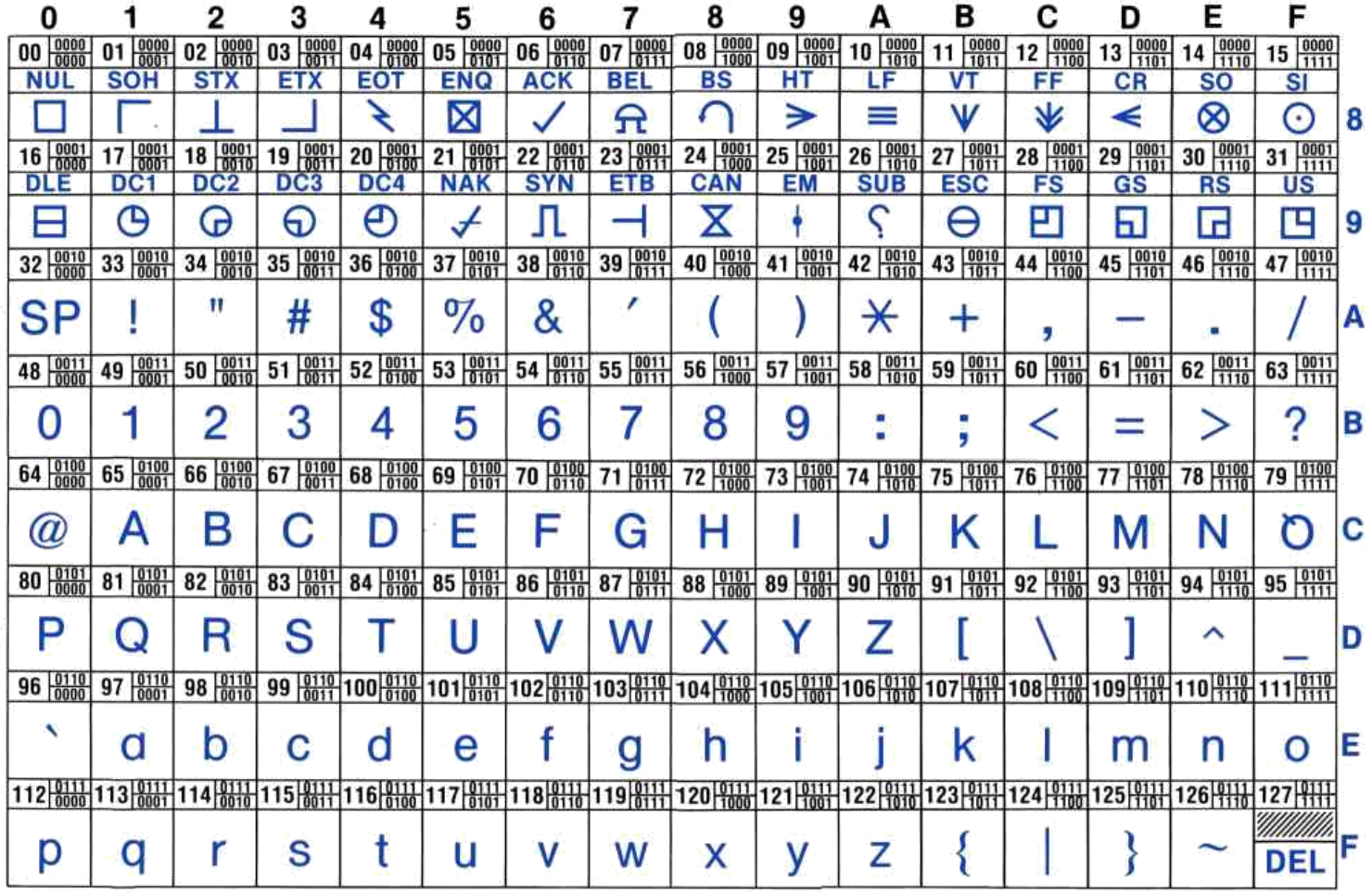
The Floating Types

- `float`, `double` and `long double` are ANSI's floating types.
- It is rare to use `long double` and many compilers simply implement it as `double`.
- A **suffix** can be appended to a floating constant to specify its type; the default is `double` which is the working floating type in C++.
- Exponent representation is acceptable, e.g., 1230(1.23e3) and 0.0003367(3.367e-4)
- A floating constant must contain either a decimal point (.), an exponential part or both.

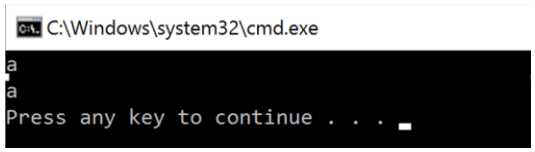
Characters & Data Type `char`

- Variables of any integral type can be used to represent characters.
- **Characters** are treated as **small integers**, and conversely, small integers are treated as characters.
- Any integral expression can be printed as a character or an integer.
- In C++, a character takes one byte or 8 bits of memory to store.

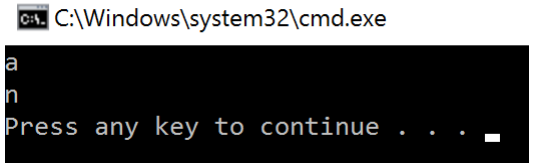
ASCII Code


0 1 2 3 4 5 6 7 8 9 A B C D E F
00 0000 0000 NUL
01 0000 0001 SOH
02 0000 0010 STX
03 0000 0011 ETX
04 0000 0100 EOT
05 0000 0101 ENQ
06 0000 0110 ACK
07 0000 0111 BEL
08 0000 1000 BS
09 0000 1001 HT
10 0000 1010 LF
11 0000 1011 VT
12 0000 1100 FF
13 0000 1101 CR
14 0000 1110 SO
15 0000 1111 SI
16 0001 0000 DLE
17 0001 0001 DC1
18 0001 0010 DC2
19 0001 0011 DC3
20 0001 0100 DC4
21 0001 0101 NAK
22 0001 0110 SYN
23 0001 0111 ETB
24 0001 1000 CAN
25 0001 1001 EM
26 0001 1010 SUB
27 0001 1011 ESC
28 0001 1100 FS
29 0001 1101 GS
30 0001 1110 RS
31 0001 1111 US
32 0010 0000 SP
33 0010 0001 !
34 0010 0010 "
35 0010 0011 #
36 0010 0100 $
37 0010 0101 %
38 0010 0110 &
39 0010 0111 ′
40 0010 1000 (
41 0010 1001)
42 0010 1010 *
43 0010 1011 +
44 0010 1100 ,
45 0010 1101 –
46 0010 1110 .
47 0010 1111 /
48 0011 0000 0
49 0011 0001 1
50 0011 0010 2
51 0011 0011 3
52 0011 0100 4
53 0011 0101 5
54 0011 0110 6
55 0011 0111 7
56 0011 1000 8
57 0011 1001 9
58 0011 1010 :
59 0011 1011 ;
60 0011 1100 <
61 0011 1101 =
62 0011 1110 >
63 0011 1111 ?
64 0100 0000 @
65 0100 0001 A
66 0100 0010 B
67 0100 0011 C
68 0100 0100 D
69 0100 0101 E
70 0100 0110 F
71 0100 0111 G
72 0100 1000 H
73 0100 1001 I
74 0100 1010 J
75 0100 1011 K
76 0100 1100 L
77 0100 1101 M
78 0100 1110 N
79 0100 1111 O
80 0101 0000 P
81 0101 0001 Q
82 0101 0010 R
83 0101 0011 S
84 0101 0100 T
85 0101 0101 U
86 0101 0110 V
87 0101 0111 W
88 0101 1000 X
89 0101 1001 Y
90 0101 1010 Z
91 0101 1011 [
92 0101 1100 \
93 0101 1101]
94 0101 1110 ^
95 0101 1111 _
96 0110 0000 `
97 0110 0001 a
98 0110 0010 b
99 0110 0011 c
100 0110 0100 d
101 0110 0101 e
102 0110 0110 f
103 0110 0111 g
104 0110 1000 h
105 0110 1001 i
106 0110 1010 j
107 0110 1011 k
108 0110 1100 l
109 0110 1101 m
110 0110 1110 n
111 0110 1111 o
112 0111 0000 p
113 0111 0001 q
114 0111 0010 r
115 0111 0011 s
116 0111 0100 t
117 0111 0101 u
118 0111 0110 v
119 0111 0111 w
120 0111 1000 x
121 0111 1001 y
122 0111 1010 z
123 0111 1011 {
124 0111 1100 |
125 0111 1101 }
126 0111 1110 ~
127 0111 1111 DEL
0 1 2 3 4 5 6 7 8 9 A B C D E F

```cpp
#include <iostream>

using namespace std;

void main()
{
    char mychac1 = 'a';
    char mychac2 = 97;

    cout << mychac1 << endl;
    cout << mychac2 << endl;

}
```




```cpp
#include <iostream>

using namespace std;

void main()
{
    char mychac1 = 'a';
    char mychac2 = 110;

    cout << mychac1 << endl;
    cout << mychac2 << endl;

}
```



Characters & Data Type `char`

Example: `char c = 'a';`

- Internally, `c` is stored as the following bit pattern
  `0 1 1 0 0 0 0 1`
- which is equivalent to a decimal `97`.

- Depending on the compiler, the type char is equivalent to either `signed char` or `unsigned char`.

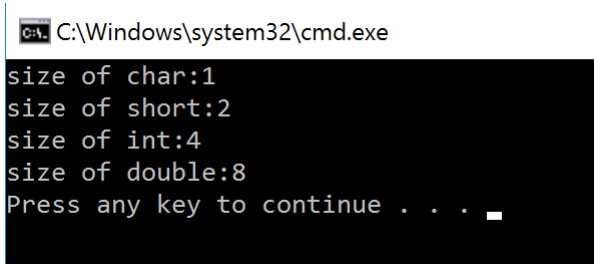
- `signed char` and `unsigned char` are used less often than `char` in typical C++ programs.

The `sizeof` Operator

- `sizeof` can be used to find the ***number of bytes needed to store an object*** (which can be a variable or a data type)
  - obviously its result is typically returned as an unsigned integer, e.g.,

```
int len1, len2;
float x;
len1 = sizeof(int);
len2 = sizeof(x);
```

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

void main()
{
    cout <<"size of char:"   <<sizeof(char) << endl;
    cout <<"size of short:"  << sizeof(short) << endl;
    cout <<"size of int:"    << sizeof(int) << endl;
    cout <<"size of double:" << sizeof(double) << endl;
}
```

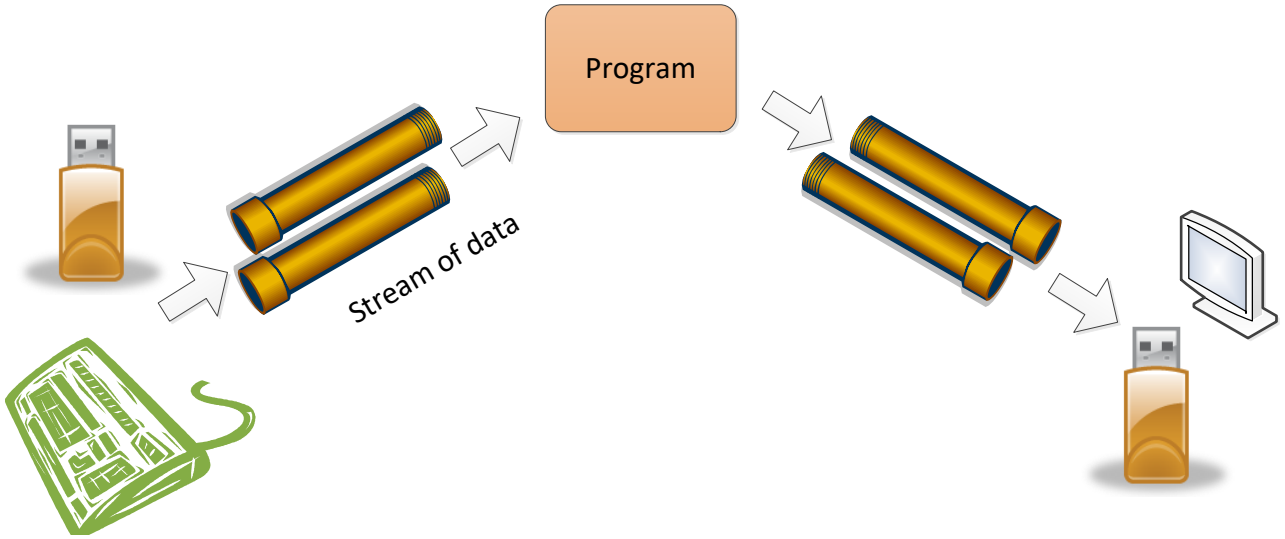


Summary-I

- Tokens
  - Keywords, Identifiers, string constants, numeric constants, operators, punctuators
- Keywords are *reserved*
- Valid and meaningful identifiers
- Variable and constant
- Global scope/local scope
- Data Type: int, float, double, unsigned…

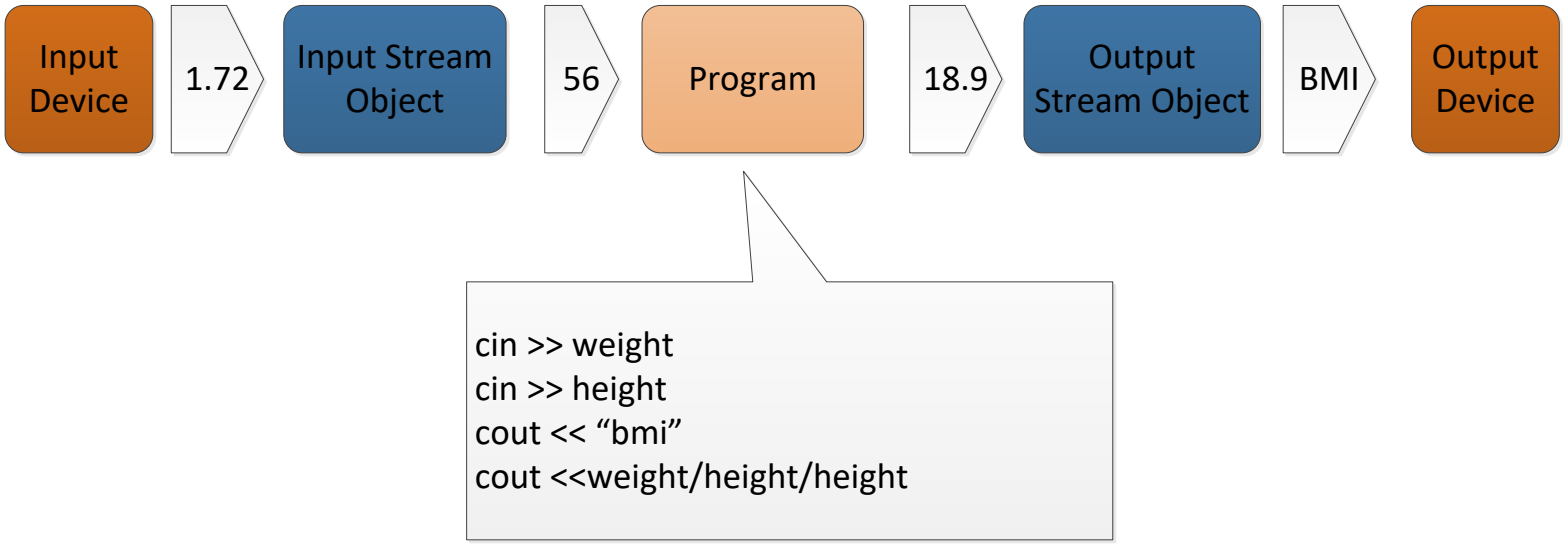
Basic I/O – Keyboard and Screen

- A program can do little if it can't take input and produce output.
- Most programs read user input from keyboard and secondary storage.
- After process the inputted data, result is commonly display on screen or write to secondary storage.



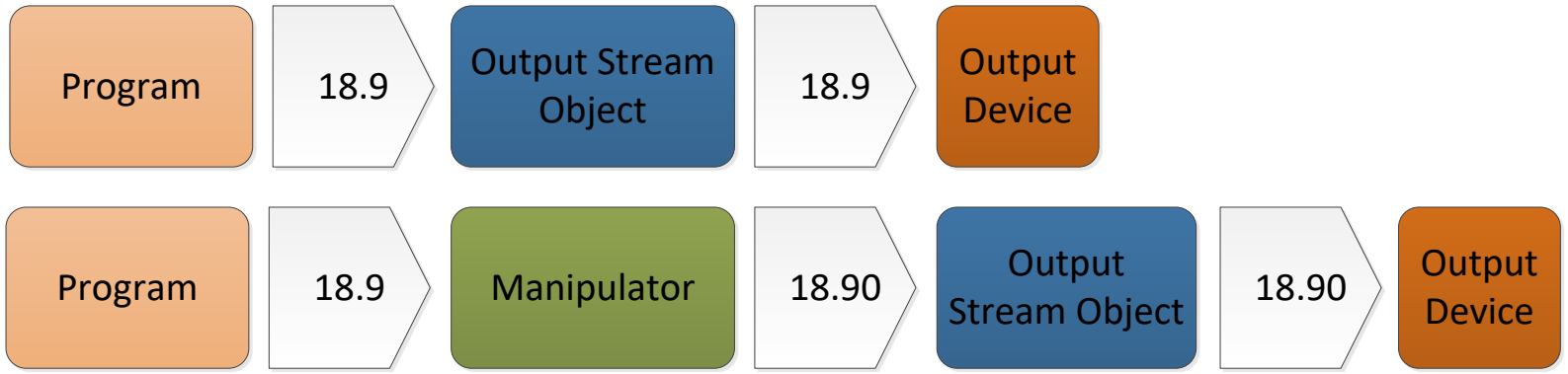
Basic I/O – `cin` and `cout`

- C++ comes with an `iostream` package (library) for basic I/O.
- *`cin`* and *`cout`* are objects defined in `iostream` for keyboard input and screen display respectively.
- To read data from `cin` and write data to `cout`, we need to use extraction operator (>>) and insertion operator (<<).



`cout`: Insertion Operator <<

- **Preprogrammed** for all standard C++ data types.
- It sends bytes to an output stream object, e.g. `cout`.
- Predefined "manipulators" can be used to change the default format of integer argument.



cout: Insertion Operator <<

Type	Expression	Output
Integer	`cout << 21`	`21`
Float	`cout << 14.5`	`14.5`
Character	`cout << 'a';` `cout <<'H' << 'i'`	`a` `Hi`
Bool	`cout << true` `cout << false`	`1` `0`
String	`cout << "hello"`	`hello`
New line (endl)	`cout << 'a' << endl << 'b';`	`a` `b`
Tab	`cout << 'a' << '\t' << 'b';`	`a  b`
Special characters	`cout << '\"' << "Hello" << '\"'` `<< endl;`	`"Hello"`
Expression	`int x=1;` `cout << 3+4 +x;`	`8`

`cout` – Change the Width of Output

Approach	Example	Output

cout.width(*width*)	`cout.width(10);`
	`cout << 5.6  << endl;`
	`cout.width(10);`
	`cout <<57.68 << endl;`
		`       5.6`
		`     57.68`

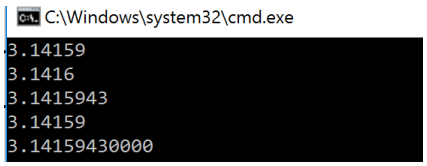
setw(*width*)	`cout << setw(5) << 1.8;`
	`cout << setw(5) << 23 <<endl;`
	`cout << setw(5) << 6.71;`
	`cout << setw(5) << 1 <<endl;`
		`  1.8   23`
		`6.71    1`

- Calling member function `width(`*`width`*`)` or using `setw` manipulator.
- Leading blanks are added to any value fewer than 10 characters width.
- If formatted output exceeds the width, the entire value prints.
- Effect lasts for one field only.

Fixed, setprecision, scientific

- Fixed
  - Set with setprecision(n)
  - Set exact as many digits (n) in the decimal part as specified by the precision field
  - No exponent part
- Scientific
  - Set with setprecision(n)
  - One digit before decimal part
  - Set exact as many digits (n) in the decimal part as specified by the precision field
  - Have the exponent part

```cpp
#include <iostream>     // std::cout, std::fixed
#include <iomanip>      // std::setprecision

int main() {
    double f = 3.1415943;
    std::cout << f << std::endl;
    std::cout << std::setprecision(5) << f << '\n';
    std::cout << std::setprecision(9) << f << '\n';
    std::cout << std::fixed;
    std::cout << std::setprecision(5) << f << '\n';
    std::cout << std::setprecision(11) << f << '\n';
    return 0;
}
```



3.14159
std::cout has a default precision of 6 -- that is, it assumes all floating point variables are only significant to 6 digits, and hence it will truncate anything after that.

3.1416
Set the number of digits (5)

3.1415943
The maximum of digits is 8 (<9)

3.14159
Fixed ->no exponent representation
Set the number of digits after decimal point (5)

3.14159430000
Fixed ->no exponent representation
Set the number of digits after decimal point (11)

`cout` – Set the Precision and format of Floating Point Output

Example	Output

`cout << setprecision(2);`
`cout << 1 <<endl;`
`cout << 1.3 <<endl;`
`cout << 1.34 <<endl;`
`cout << 0.0000000134 << endl;`
`cout << fixed;`
`cout << 0.0000000134 << endl;`
`cout << scientific << 0.0005 << endl;`
	`1`
	`1.3`
	`1.3`
	`1.3e-008`
	`0.00`
	`5.00e-004`

- Must `#include <iomanip>.`
- Floating-point precision is six by default.
- Use `setprecision`, `fixed` and `scientific` manipulators to change the precision value and printing format.
- Effect is permanent.

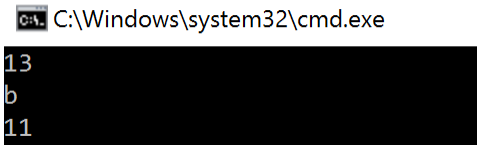
cout – Other Manipulators

Manipulators	Example	Output

fill	`cout << setfill('*');`
	`cout << setw(10);`
	`cout << 5.6  << endl;`
	`cout << setw(10);`
	`cout <<57.68 << endl;`
		`*******5.6`
		`*****57.68`

radix	`cout << oct << 11 << endl;`
	`cout << hex << 11 << endl;`
	`cout << dec << 11 << endl;`
	`(oct: base 8, hex: base 16)`
		13
		b
		11

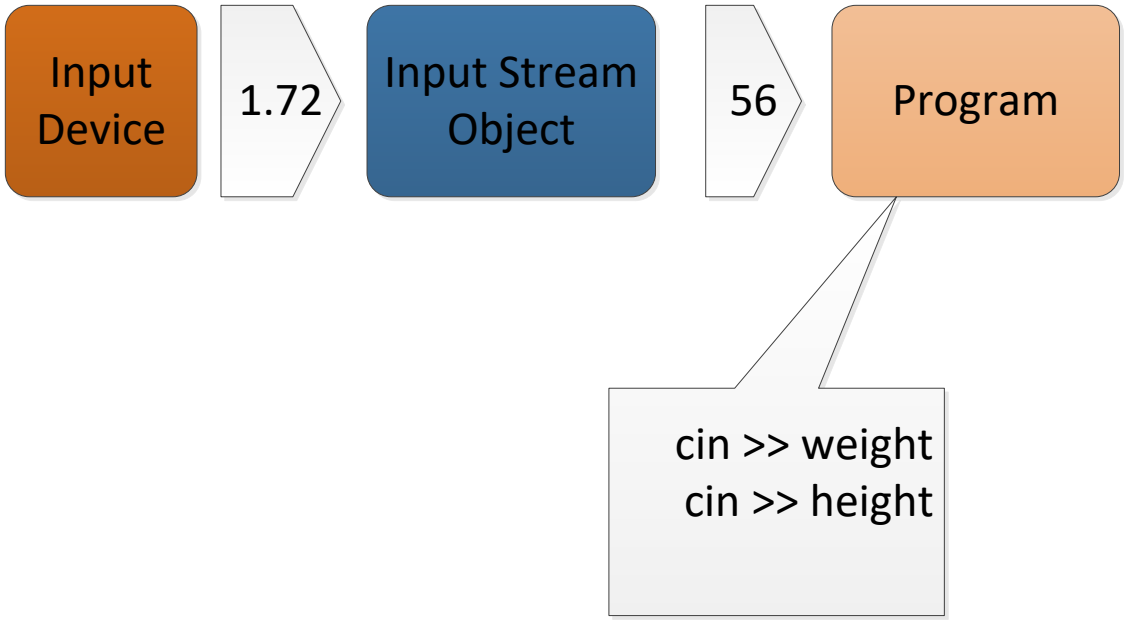
```
#include <iostream>
#include <iomanip>
using namespace std;

void main()
{
    cout << oct << 11 << endl;
    cout << hex << 11 << endl;
    cout << dec << 11 << endl;
}
```



`cin`: Extraction Operator >>

Preprogrammed for all standard C++ data types.

- Get bytes from an **input stream** object.
- Depend on **white space** to separate incoming data values.

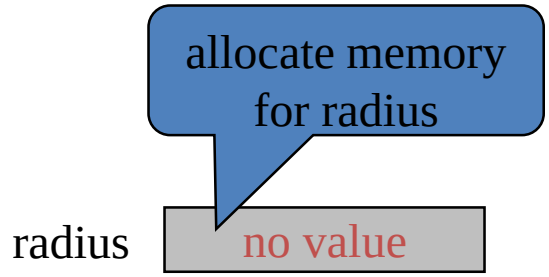


Extraction Operator

Type	Variable	Expression	Input	x	y
Integer	`int x,y;`	`cin >> x;`	`21`	`21`	
		`cin >> x >> y;`	`5 3`	`5`	`3`
Float	`float x,y;`	`cin >> x;`	`14.5`	`14.5`	
Character	`char x,y;`	`cin >> x;`	`a` `Hi`	`a` `H`	
		`cin >> x >> y;`	`Hi`	`H`	`i`
String	`char x[20];` `char y[20];`	`cin >> x;`	`hello`	`hello`	
		`cin >> x >> y`	`Hello` `World`	`Hello`	`World`

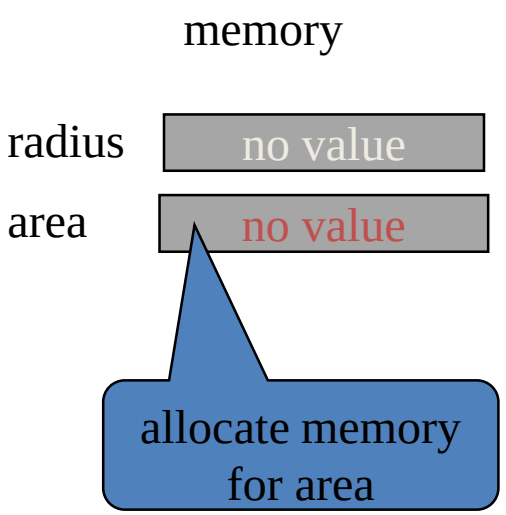
Trace a Program Execution

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

int main() {
  double radius;
  double area;

  // Step 1: Read in radius
  radius = 20;

  // Step 2: Compute area
  area = radius * radius * 3.14159;

  // Step 3: Display the area
  cout << "The area is ";
  cout << area << endl;
}
```



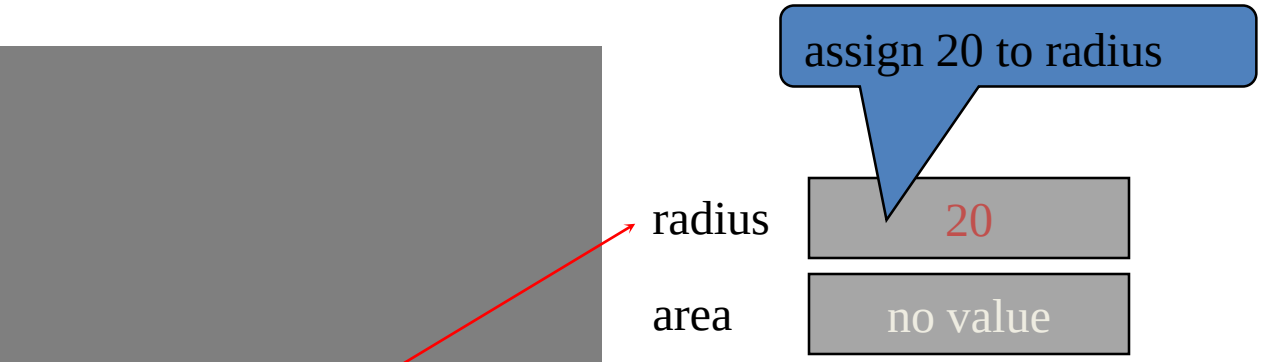
Trace a Program Execution

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

int main() {
  double radius;
  double area;

  // Step 1: Read in radius
  radius = 20;

  // Step 2: Compute area
  area = radius * radius * 3.14159;

  // Step 3: Display the area
  cout << "The area is ";
  cout << area << std::endl;
}
```



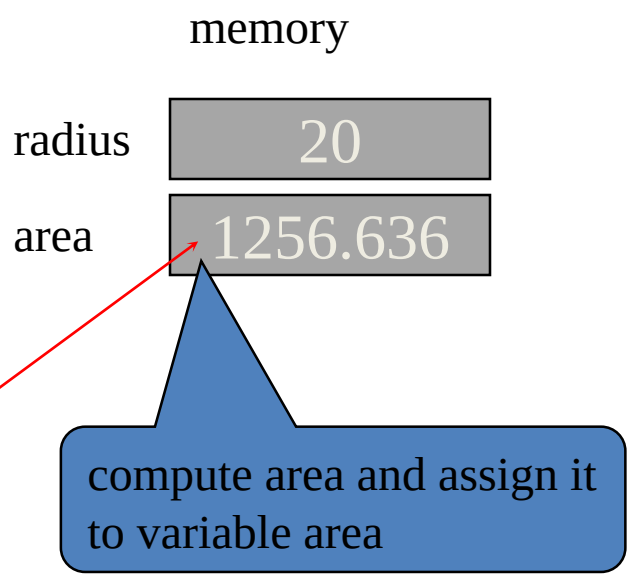
Trace a Program Execution

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

int main() {
  double radius;
  double area;

  // Step 1: Read in radius
  radius = 20;

  // Step 2: Compute area
  area = radius * radius * 3.14159;

  // Step 3: Display the area
  cout << "The area is ";
  cout << area << std::endl;
}
```



Trace a Program Execution

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

int main() {
  double radius;
  double area;

  // Step 1: Read in radius
  radius = 20;

  // Step 2: Compute area
  area = radius * radius * 3.14159;

  // Step 3: Display the area
  cout << "The area is ";
  cout << area << std::endl;
}
```



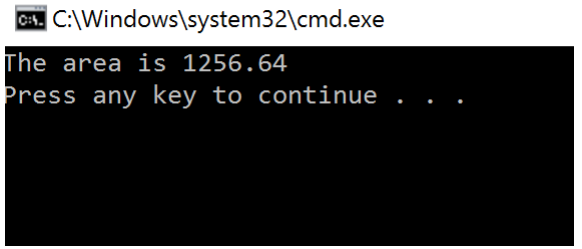
Trace a Program Execution

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

int main() {
  double radius;
  double area;

  // Step 1: Read in radius
  radius = 20;

  // Step 2: Compute area
  area = radius * radius * 3.14159;

  // Step 3: Display the area
  cout << "The area is ";
  cout << area << std::endl;
}
```

memory

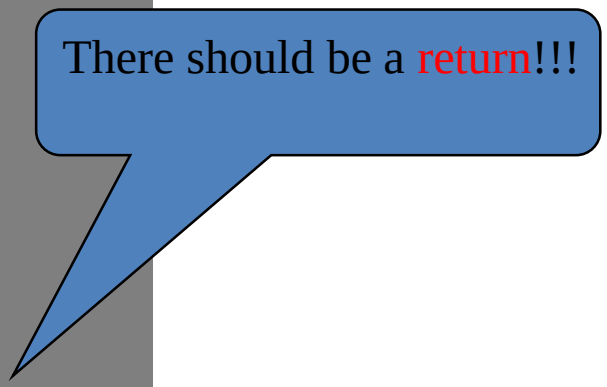
radius	20
area	1256.636

print a message to the console



Trace a Program Execution

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

int main() {
  double radius;
  double area;

  // Step 1: Read in radius
  radius = 20;

  // Step 2: Compute area
  area = radius * radius * 3.14159;

  // Step 3: Display the area
  cout << "The area is ";
  cout << area << std::endl;
}
```



#include <iomanip>

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

int main()
{
    cout << "12345678901234567890" << endl;
    cout << "====================" << endl;
    c                              < setw(10) << "Hello" << 3 << endl;
    c                                 setw(4) << 67;
    cout                 setw(5) << 45 << setfill(' ') << setw(6) << 23 << endl;


}
```

There should be a return!!!

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

int main()
{
    cout << "12345678901234567890" << endl;
    cout << "====================" << endl;
    cout << setw(4) << 89 << setw(10) << "Hello" << 3 << endl;
    cout << setfill('0') << setw(4) << 67;
    cout << setw(5) << 45 << setfill(' ') << setw(6) << 23 << endl;

    return 0;
}
```

```
1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0
= = = = = = = = = = = = = = = = = = = =
    8 9             H e l l o 3
0 0 6 7 0 0 0 4 5               2 3
```

Summary

- Most computer programs access data during its execution. Data are stored in main memory as **variable** and **constant**.
- Variables and constant are referred by their **identifiers**.
- In C++, variable and constant must be **typed**.
- The place when variable/constant is defined determined its **scope** and hence determined its accessibility.
- **`cin`** and **`cout`** are objects defined in `iostream`. They represent the keyboard and screen display respectively.
- program uses
  - extraction operator **>>** to read input from `cin`.
  - Insertion operator **<<** to write output to `cout`.
- **Manipulator** can be added to `cout` for output formating.

FAQ

- Side effects of global variables
- Can global variables be declared in the middle?
- Fixed, setprecision, scientific representations
- Revision

Side effects of global variables

- Any function using global variables-instead of passing variables
  - Not independent

- Global variables allow the programmers to "jump around" the normal safeguards provides by functions
  - Do not make all variables global

Can global variables be declared in the middle?

Yes, but variable and constant must be declared before use.

```
#include <iostream>
using namespace std;
int calcmin();

int main()
{
    cout << calcmin()<<endl;
    return 1;
}


int number1=2;
int number2=4;


int calcmin()
{
    if (number1 > number2)
    {
        return number2;
    }
    else
    {
        return number1;
    }
}
```