V2
2018

មេរៀនទី ២

C++ Basic Syntax

ក្នុងមេរៀននេះយើងនឹងសិក្សាអំពី :

❖ Program Structure , Compile and Execute, Semicolons and Blocks, Identifiers, Keywords, Trigraphs , Whitespace and Comments.

C++ Programming
រៀនដើម្បីភាពជោគជ័យ!

រៀបចំដោយ : សេង ស៊ង់
Tel : 092 /093 77 12 44
Site : www.sourngonline.com

SourngOnlineTutorial   fb.com/SourngOnlineTutorial   tw.com/sourng2009   092771244

២.១ សេចក្តីផ្តើម

- នៅពេលយើងពិចារណាកម្មវិធី C ++ វាអាចត្រូវបានកំនត់ថាជាការប្រមូលផ្តុំវត្ថុដែលទាក់ទងតាមរយៈវិធីសាស្រ្តរបស់គ្នាទៅវិញទៅមក។ ឥឡូវនេះសូមឱ្យយើងពិនិត្យមើលដោយសង្ខេបនូវអត្ថន័យនៃ class, object, methods, and instant variables ថាវាជាអ្វី។
  - ❑ វត្ថុ(object) - វត្ថុមានរដ្ឋនិងអាកប្បកិរិយា។ ឧទាហរណ៍: សត្វឆ្កែមួយមានពណ៌ ដូចជាឈ្មោះ ពូជ និងឥរិយាបថ - ការរអ៊ូរទាំ សំបក និងការញ៉ាំ។ វត្ថុមួយគឺជាវត្ថុមួយនៃថ្នាក់។

២.១ សេចក្ដីផ្ដើម

- ❑ថ្នាក់(class) - ថ្នាក់មួយអាចត្រូវបានកំណត់ជាគំរូ / ប្លង់មេដែលពិពណ៌នាអំពីអាកប្បកិរិយា / ស្ថានភាពដែលទ្រទ្រង់ប្រភេទរបស់វា។
- ❑វិធីសាស្ត្រ(Methods) - វិធីសាស្ត្រគឺជាឥរិយាបថជាមូលដ្ឋានមួយ។ ថ្នាក់មួយអាចមានវិធីសាស្ត្រជាច្រើន។ វាស្ថិតនៅក្នុងវិធីសាស្ត្រដែលតក្កវិទ្យាត្រូវបានសរសេរទិន្នន័យត្រូវបានរៀបចំ ហើយសកម្មភាពទាំងអស់ត្រូវបានប្រតិបត្តិ។
- ❑អថេរវត្ថុ(Instance Variables) - វត្ថុនីមួយៗមានសំណុំអថេរវត្ថុតែមួយគត់របស់វា។ ស្ថានភាពរបស់វត្ថុត្រូវបានបង្កើតដោយតម្លៃដែលបានផ្ដល់ទៅអថេរវត្ថុទាំងនេះ។

២.២ រចនាសម្ព័ន្ធកម្មវិធី

- ចូរយើងក្រឡេកមើលកូដដ៏សាមញ្ញមួយដែលនឹងបោះពុម្ពពាក្យ Hello World ។

```
#include <iostream>
using namespace std;
// main() is where program execution begins.
 int main() {
     cout << "Hello World"; // prints Hello World
     return 0;

}
```

២.២ រចនាសម្ព័ន្ធកម្មវិធី

- ចូរយើងមើលផ្នែកផ្សេងៗនៃកម្មវិធីខាងលើ
  - ➢ភាសា C ++ កំណត់បឋមកថាជាច្រើនដែលមានព័ត៌មានដែលចាំបាច់ ឬមានប្រយោជន៍ចំពោះកម្មវិធីរបស់អ្នក។ សម្រាប់កម្មវិធីនេះ, បឋមកថា <iostream> គឺត្រូវបានត្រូវការ។
  - ➢បន្ទាត់ដោយប្រើ namespace std; ប្រាប់កម្មវិធីចងក្រងប្រើឈ្មោះគន្លឹះ std ។ ចន្លោះឈ្មោះគឺជាការបន្ថែមថ្មីៗទៅនឹង C ++ ។

២.២ រចនាសម្ព័ន្ធកម្មវិធី

- បន្ទាត់បន្ទាប់ '// main() is where program execution begins.' គឺជាសេចក្ដីអធិប្បាយបន្ទាត់តែមួយដែលមាននៅក្នុង C ++ ។ យោបល់បន្ទាត់តែមួយចាប់ ផ្ដើមដោយ // ហើយឈប់នៅចុងបន្ទាត់។
- បន្ទាត់ int main () គឺជាមុខងារចម្បងដែលចាប់ផ្ដើមដំណើរការកម្មវិធី។
- បន្ទាត់បន្ទាប់ cout << "Hello World"; បណ្ដាលឱ្យ សារ " Hello World " ដែលត្រូវបង្ហាញនៅលើអេក្រង់។
- បន្ទាត់បន្ទាប់ return 0; បញ្ចប់មុខងារ main( )function ហើយបណ្ដាលឱ្យវាត្រឡប់តម្លៃ 0 ទៅដំណើរការហៅ។

២.៣ Compile and Execute C++ Program

❖តោះមើលរបៀបរក្សាទុកឯកសារចងក្រងនិងដំណើរការកម្មវិធី។ សូមអនុវត្តតាមជំហានដូចខាងក្រោម:

- បើកកម្មវិធីកែសម្រួលអត្ថបទហើយបន្ថែមកូដដូចខាងលើ។
- រក្សាទុកឯកសារជា: hello.cpp
- បើកប្រអប់បញ្ចូលពាក្យបញ្ជាហើយចូលទៅថតដែលអ្នកបានរក្សាទុក។
- វាយ 'g++ hello.cpp' ហើយចុចបញ្ចូលដើម្បីចងក្រងកូដរបស់អ្នក។ ប្រសិនបើគ្មានកំហុសក្នុងកូដរបស់អ្នកទេពាក្យបញ្ជានឹងនាំអ្នកទៅកាន់បន្ទាត់បន្ទាប់ហើយបង្កើតឯកសារដែលអាចប្រតិបត្តិបាន a.out ។
- ឥឡូវវាយ 'a.out' ដើម្បីដំណើរការកម្មវិធីរបស់អ្នក។
- អ្នកនឹងអាចមើលឃើញ "សួស្តីពិភពលោក" បានបោះពុម្ពនៅលើបង្អួច។

២.៣ Compile and Execute C++ Program

```
$ g++ hello.cpp
$ ./a.out
Hello World
```

❖Make sure that g++ is in your path and that you are running it in the directory containing file hello.cpp.

២.៤ Semicolons and Blocks in C++

ត

- នៅក្នុង C ++ សញ្ញាចុចគឺសញ្ញាបញ្ចប់។ នោះមានន័យថាstatementsនិមួយៗត្រូវតែបញ្ចប់ដោយមានសញ្ញា Semicolons ។ វាចង្អុលបង្ហាញពីការបញ្ចប់នៃធាតុឡូជីខលមួយ។
- ឧទាហរណ៏ខាងក្រោមនេះគឺជាstatementsខុសៗគ្នាបី។

```
x = y;
y = y + 1;
add(x, y);
```

២.៤ Semicolons and Blocks in C++

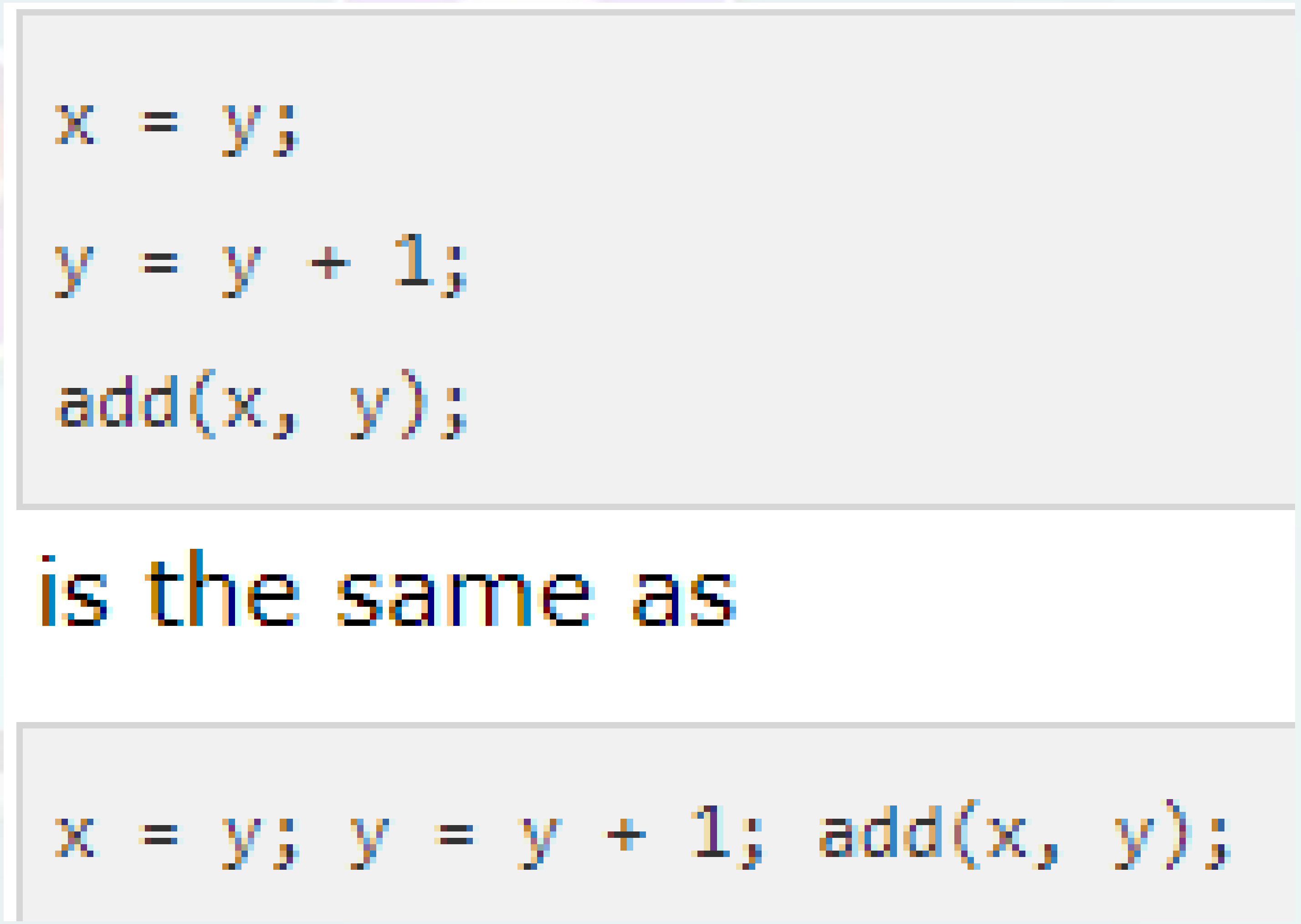
- ប្លុកគឺជាសំណុំនៃstatements តភ្ជាប់ដែលត្រូវបានហ៊ុំព័ទ្ធដោយបើកនិងបិទដង្កៀប។ ឧទាហរណ៍ -

```
{
   cout << "Hello World"; // prints Hello World
   return 0;
}
```

២.៤ Semicolons and Blocks in C++

C ++ មិនស្គាល់ចុងបញ្ចប់នៃបន្ទាត់ជាចុង។ សម្រាប់ហេតុផលនេះវាមិនមានបញ្ហាដែលអ្នកដាក់statements នៅក្នុងបន្ទាត់។

ឧទាហរណ៍ -ដូចក្នុងរូប



```
x = y;

y = y + 1;

add(x, y);
```

is the same as

```
x = y; y = y + 1; add(x, y);
```

២.៥ C++ Identifier

- A C++ identifier is a name used to identify a variable, function, class, module, or any other user-defined item. An identifier starts with a letter A to Z or a to z or an underscore (_) followed by zero or more letters, underscores, and digits (0 to 9).
- C++ does not allow punctuation characters such as @, $, and % within identifiers. C++ is a case-sensitive programming language.
  Thus, Manpower and manpower are two different identifiers in C++.

២.៥ C++ Identifier

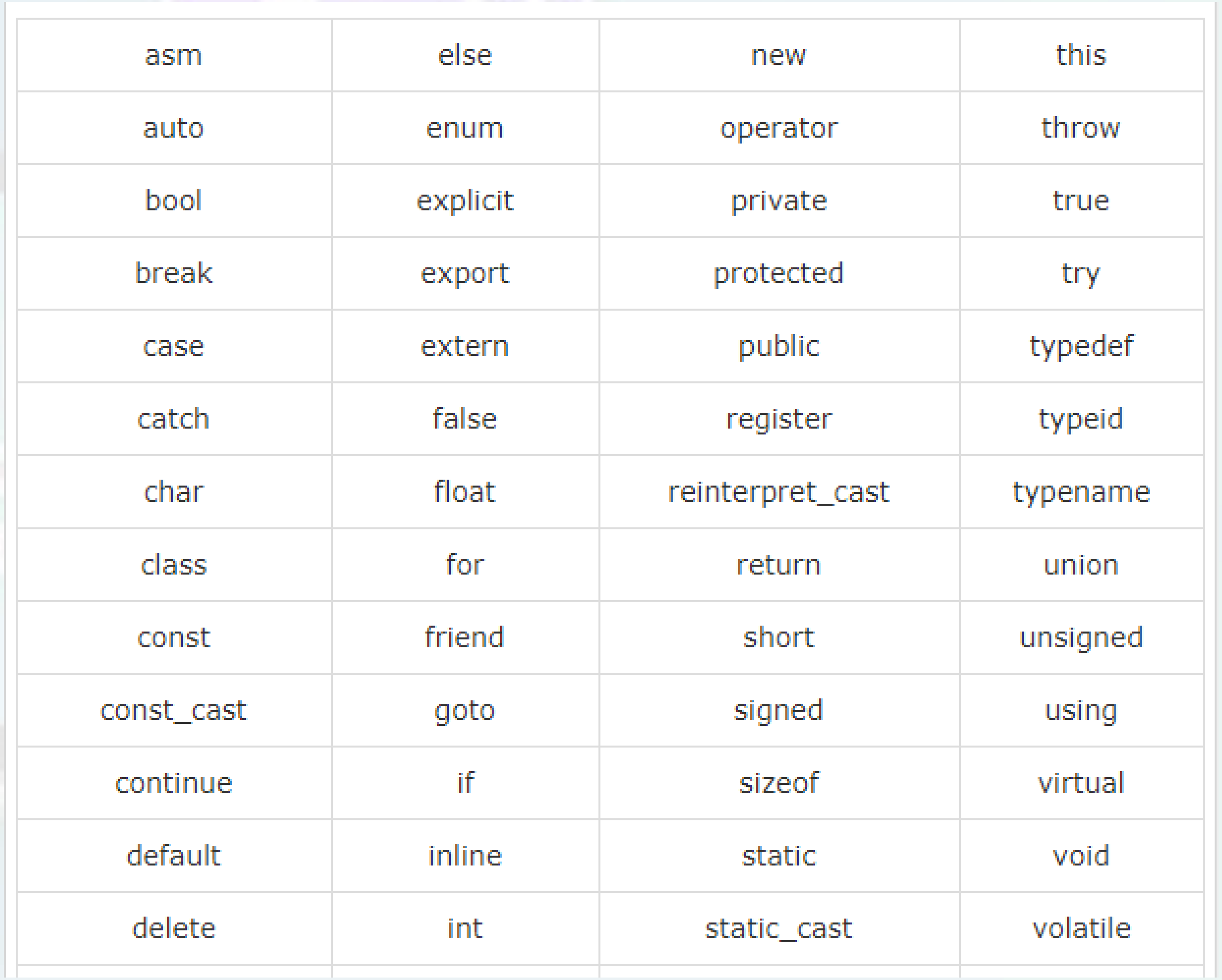
ត

- Here are some examples of acceptable identifiers –

```
mohd        zara    abc   move_name  a_123

myname50   _temp  j     a23b9      retVal
```

២.៦ C++ Keywords

- The following list shows the reserved words in C++. These reserved words may not be used as constant or variable or any other identifier names.



asm	else	new	this
auto	enum	operator	throw
bool	explicit	private	true
break	export	protected	try
case	extern	public	typedef
catch	false	register	typeid
char	float	reinterpret_cast	typename
class	for	return	union
const	friend	short	unsigned
const_cast	goto	signed	using
continue	if	sizeof	virtual
default	inline	static	void
delete	int	static_cast	volatile

V2
2018

មេរៀនទី 3

Objects, types, and values

C++ Programming
រៀនដើម្បីភាពជោគជ័យ!

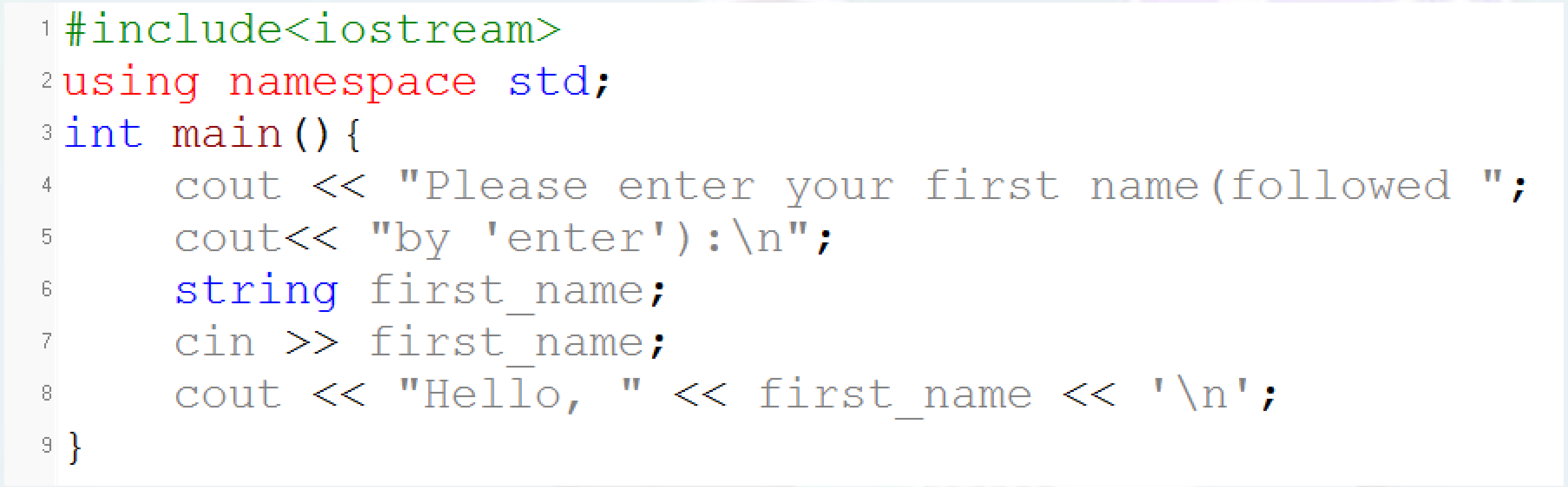
រៀបចំដោយ : សេង ស៊ង់
Tel : 092 /093 77 12 44
Site : www.sourngonline.com

SourngOnlineTutorial fb.com/SourngOnlineTutorial tw.com/sourng2009 092771244

3.1 Overview

- Strings and string I/O
- Integers and integer I/O
- Types and objects
- Type safety

3.2. Input and output



```
#include<iostream>
using namespace std;
int main(){
     cout << "Please enter your first name(followed ";
     cout<< "by 'enter'):\n";
     string first_name;
     cin >> first_name;
     cout << "Hello, " << first_name << '\n';
}
```

// note how several values can be output by a single statement

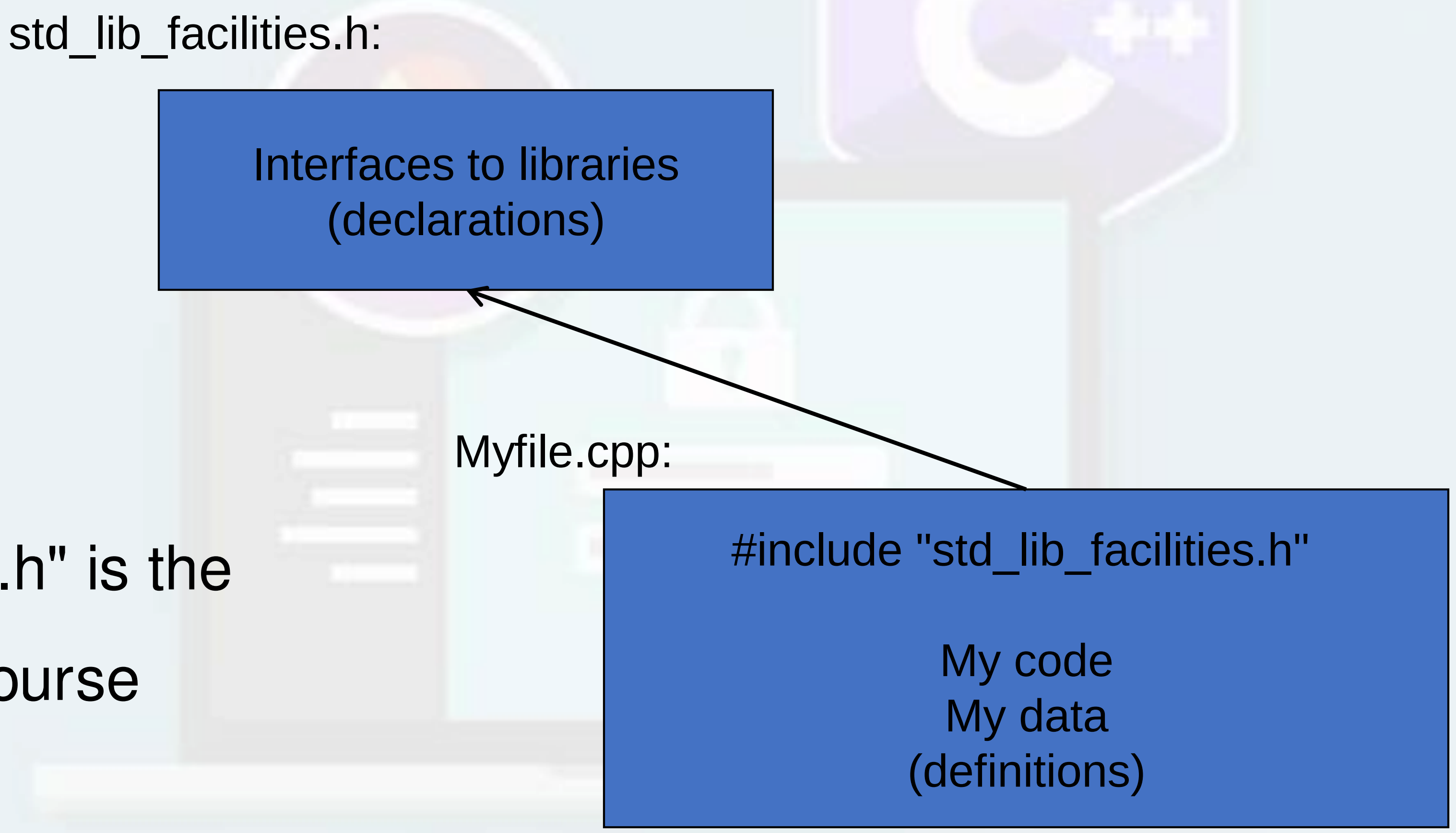
// a statement that introduces a variable is called a declaration

// a variable holds a value of a specified type

*// the final **return 0;** is optional in **main()***

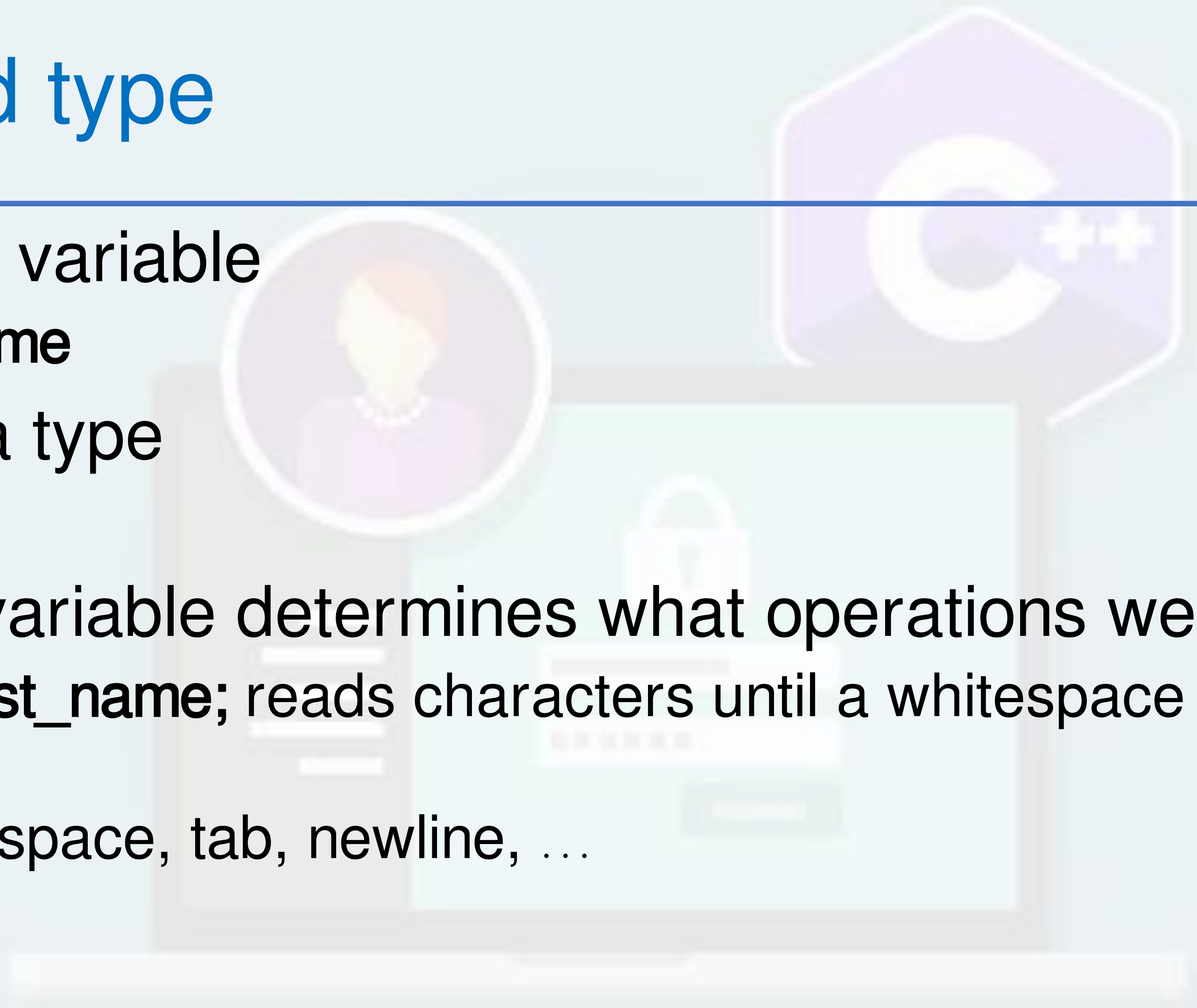
// but you may need to include it to pacify your compiler

3.3 Source files


std_lib_facilities.h:
Interfaces to libraries (declarations)
Myfile.cpp:
#include "std_lib_facilities.h"
My code
My data
(definitions)

- "std_lib_facilities.h" is the header for our course

3.4 Input and type



- We read into a variable
  - Here, **first_name**
- A variable has a type
  - Here, **string**
- The type of a variable determines what operations we can do on it
  - Here, **cin>>first_name;** reads characters until a whitespace character is seen ("a word")
  - White space: space, tab, newline, …

3.5 String input

```
// read first and second name:
int main()
{
   cout << "please enter your first and second names\n";
   string first;
   string second;
   cin >> first >> second;                    // read two strings
   string name = first + ' ' + second;        // concatenate strings
                                              // separated by a space
   cout << "Hello, "<< name << '\n';
}

// I left out the #include  "std_lib_facilities.h" to save space and
//  reduce distraction
// Don't forget it in real code
// Similarly, I left out the Windows-specific keep_window_open();
```

3.6 ចំនួនគត់ (Integers)

```
// read name and age:
int main()
{
   cout << "please enter your first name and age\n";
   string first_name;             // string variable
   int age;                       // integer variable
   cin >> first_name >> age;      // read
   cout << "Hello, " << first_name << " age " << age << '\n';
}
```

3.7 Integers and Strings

- Strings
  - **cin >>** reads a word
  - **cout <<** writes
  - **+** concatenates
  - **+= s** adds the string **s** at end
  - **++** is an error
  - **-** is an error
  - . . .

- Integers and floating-point numbers
  - **cin >>** reads a number
  - **cout <<** writes
  - **+** adds
  - **+= n** increments by the int **n**
  - **++** increments by **1**
  - **-** subtracts
  - . . .

The type of a variable determines which operations are valid and what their meanings are for that type (that's called "overloading" or "operator overloading")

3.8 ការដាក់ឈ្មោះ (Names)

- A name in a C++ program
  - Starts with a letter, contains letters, digits, and underscores (only)
    - x, number_of_elements, Fourier_transform, z2
    - Not names:
      - 12x
      - time$to$market
      - main line
    - Do not start names with underscores: _foo
      - those are reserved for implementation and systems entities

- Users can't define names that are taken as keywords
  - E.g.:
    - int
    - if
    - while
    - double

3.8 ការដាក់ឈ្មោះ (Names)

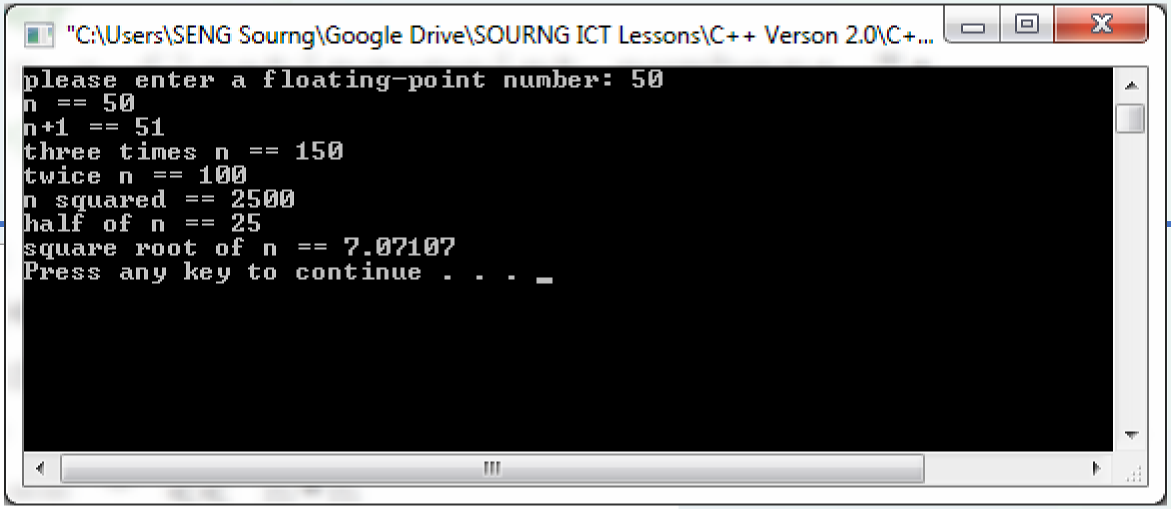
- Choose meaningful names
  - Abbreviations and acronyms can confuse people
    - **mtbf, TLA, myw, nbv**
  - Short names can be meaningful
    - (only) when used conventionally:
      - x is a local variable
      - i is a loop index

3.8 ការដាក់ឈ្មោះ (Names)

- Don't use overly long names
  - Ok:
    - partial_sum
      element_count
      staple_partition
  - Too long:
    - the_number_of_elements
      remaining_free_slots_in_the_symbol_table

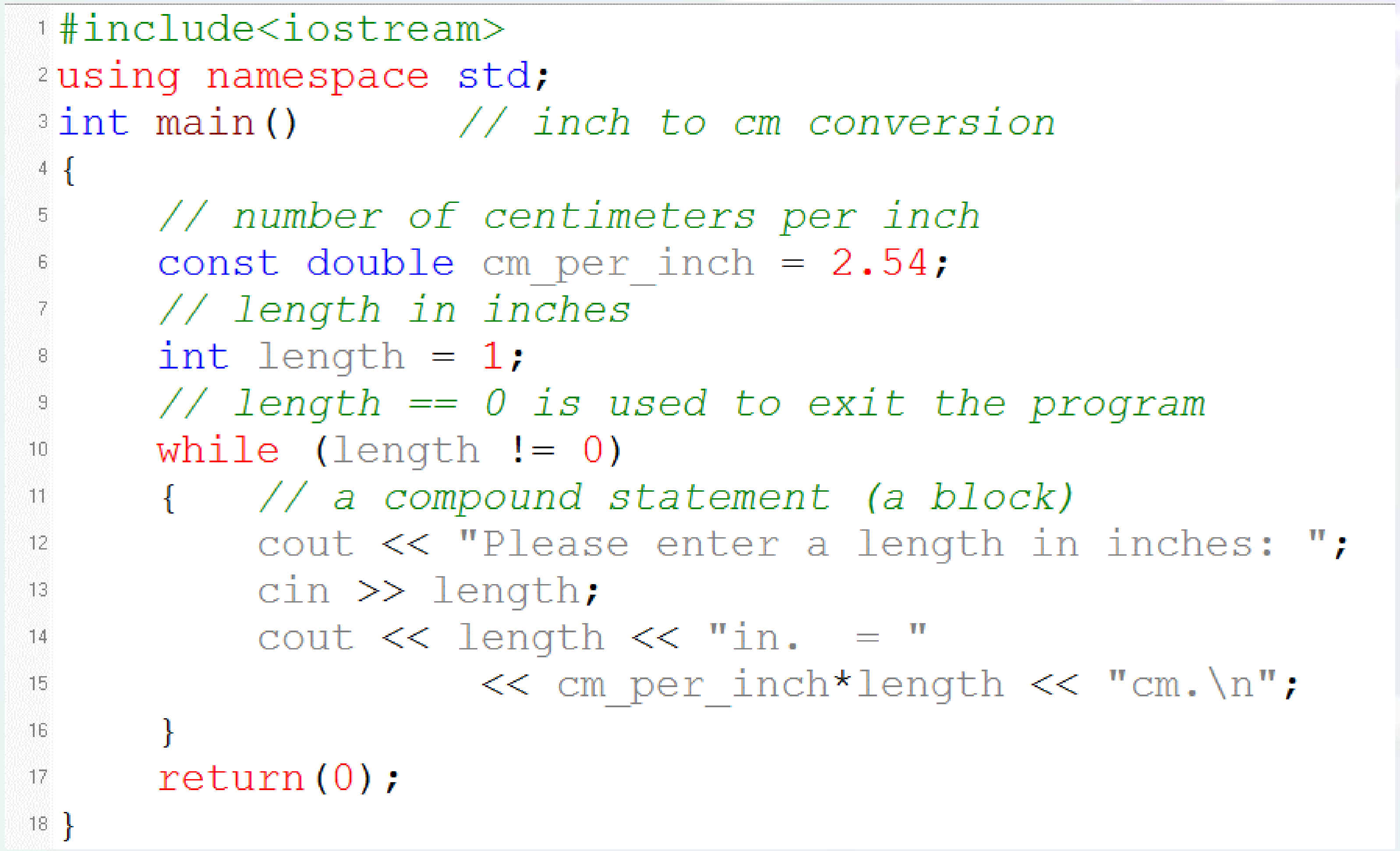
3.9 Simple arithmetic

```cpp
//do a bit of very simple arithmetic:
#include<iostream>
#include<math.h>
using namespace std;
int main()
{    // prompt for a number
     cout << "please enter a floating-point number: ";
     double n; // floating-point variable
     cin >> n;
     cout << "n== "<< n
          << "\nn+1 =="<<n+1      // '\n' means “a newline”
          << "\nthree times n =="<<3*n
          << "\ntwice n == " << n+n
          << "\nn squared == " << n*n
          << "\nhalf of n == " << n/2
          << "\nsquare root of n == "<<sqrt(n)// library function
          << "\n";
}
```


"C:\Users\SENG Sourng\Google Drive\SOURNG ICT Lessons\C++ Verson 2.0\C+...

```
please enter a floating-point number: 50
n == 50
n+1 == 51
three times n == 150
twice n == 100
n squared == 2500
half of n == 25
square root of n == 7.07107
Press any key to continue . . .
```

3.10 A simple computation



```cpp
#include<iostream>
using namespace std;
int main()      // inch to cm conversion
{
    // number of centimeters per inch
    const double cm_per_inch = 2.54;
    // length in inches
    int length = 1;
    // length == 0 is used to exit the program
    while (length != 0)
    {   // a compound statement (a block)
        cout << "Please enter a length in inches: ";
        cin >> length;
        cout << length << "in.  = "
                 << cm_per_inch*length << "cm.\n";
    }
    return(0);
}
```

A while-statement repeatedly executes until its condition becomes false

3.11 Types and literals

- Built-in types
  - Boolean type
    - **bool**
  - Character types
    - **char**
  - Integer types
    - **int**
      - **and short** and **long**
  - Floating-point types
    - **double**
      - and **float**
- Standard-library types
  - **string**
  - **complex<Scalar>**

- Boolean literals
  - true false
- Character literals
  - 'a', 'x', '4', '\n', '$'
- Integer literals
  - 0, 1, 123, -6, 034, 0xa3
- Floating point literals
  - 1.2, 13.345, .3, -0.54, 1.2e3, .3F
- String literals **"asdf"**,

  "Howdy, all y'all!"
- Complex literals
  - complex<double>(12.3,99)
  - complex<float>(1.3F)

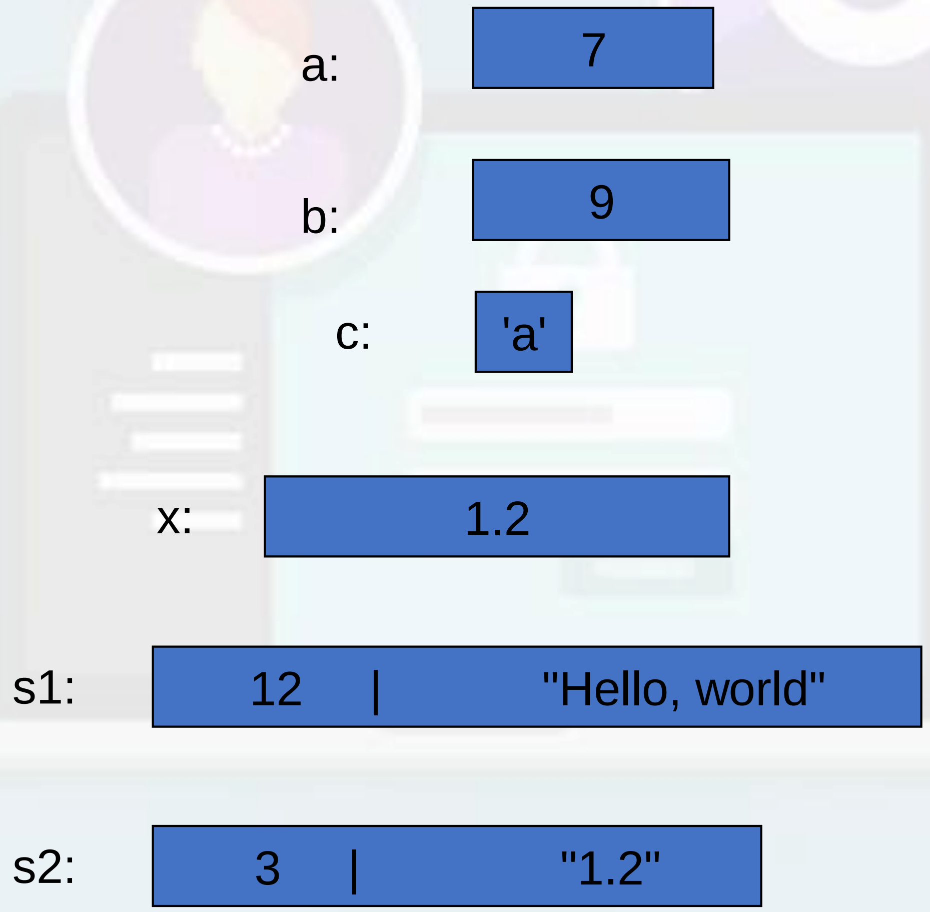
If (and only if) you need more details, see the book!

3.12 Types

- C++ provides a set of types
  - E.g. **bool**, **char**, **int**, **double**
  - Called "built-in types"
- C++ programmers can define new types
  - Called "user-defined types"
  - We'll get to that eventually
- The C++ standard library provides a set of types
  - E.g. **string**, **vector**, **complex**
  - Technically, these are user-defined types
    - they are built using only facilities available to every user

3.13 Declaration and initialization

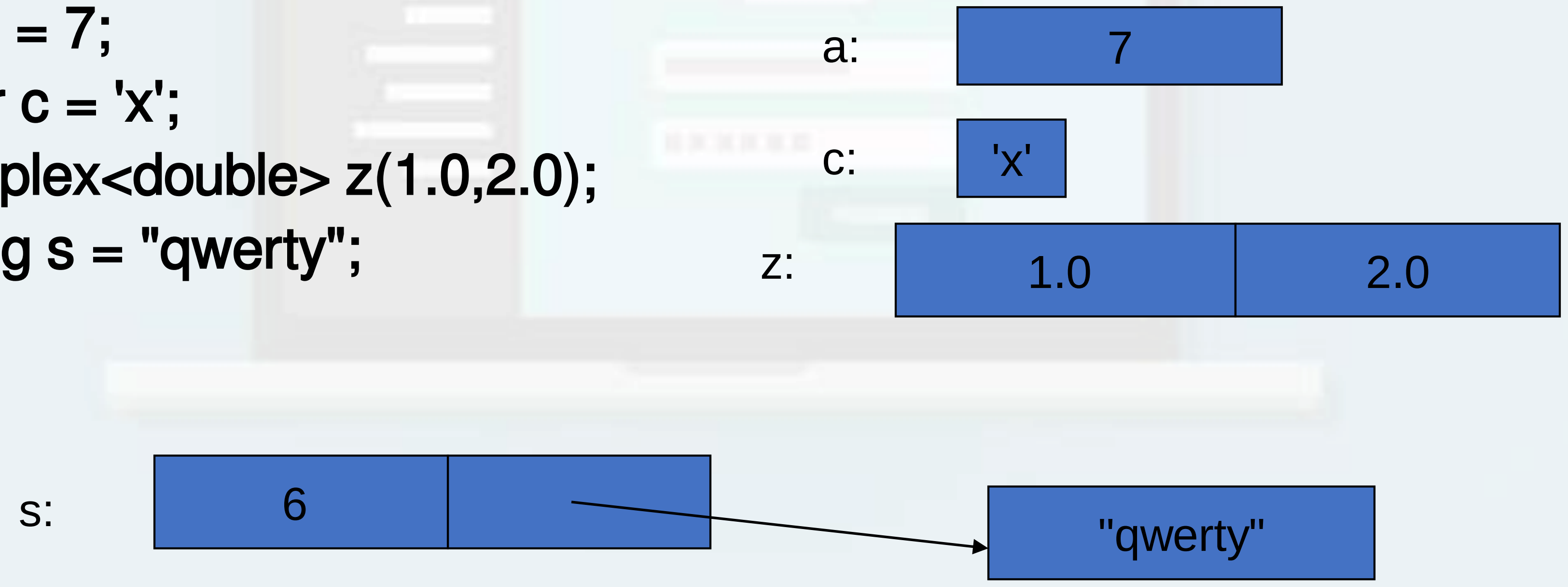
```
int a = 7;

int b = 9;

char c = 'a';

double x = 1.2;

string s1 = "Hello, world";

string s2 = "1.2";
```



3.14 Objects

- An object is some memory that can hold a value of a given type
- A variable is a named object
- A declaration names an object

```
int a = 7;
char c = 'x';
complex<double> z(1.0,2.0);
string s = "qwerty";
```



3.15 Type safety

- Language rule: type safety
  - Every object will be used only according to its type
    - A variable will be used only after it has been initialized
    - Only operations defined for the variable's declared type will be applied
    - Every operation defined for a variable leaves the variable with a valid value

3.16 Type safety

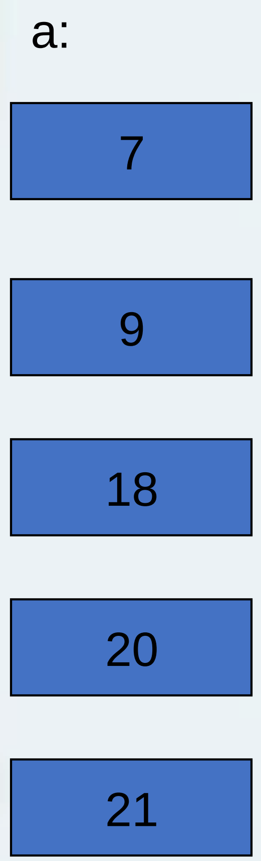
- Ideal: static type safety
  - A program that violates type safety will not compile
    - The compiler reports every violation (in an ideal system)
- Ideal: dynamic type safety
  - If you write a program that violates type safety it will be detected at run time
    - Some code (typically "the run-time system") detects every violation not found by the compiler (in an ideal system)

3.16 Type safety

- Type safety is a very big deal
  - Try very hard not to violate it
  - “when you program, the compiler is your best friend”
    - But it won´t feel like that when it rejects code you’re sure is correct
- C++ is not (completely) statically type safe
  - No widely-used language is (completely) statically type safe
  - Being completely statically type safe may interfere with your ability to express ideas
- C++ is not (completely) dynamically type safe
  - Many languages are dynamically type safe
  - Being completely dynamically type safe may interfere with the ability to express ideas and often makes generated code bigger and/or slower
- Almost all of what you´ll be taught here is type safe
  - We´ll specifically mention anything that is not

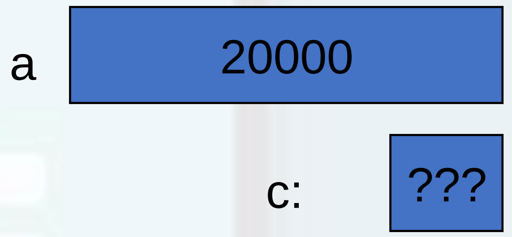
3.17 Assignment and increment

```
// changing the value of a variable

int a = 7;        // a variable of type int called a

                     // initialized to the integer value 7

a = 9;               // assignment: now change a's value to 9


a = a+a;          // assignment: now double a's value


a += 2;          // increment a's value by 2


++a;                // increment a's value (by 1)
```



3.17 A type-safety violation (“implicit narrowing”)

```
// Beware: C++ does not prevent you from trying to put a large value
// into a small variable (though a compiler may warn)

int main()
{
    int a = 20000;
    char c = a;
    int b = c;
    if (a != b)                         //  != means "not equal"
            cout << "oops!: " << a << "!=" << b << '\n';
    else
            cout << "Wow! We have large characters\n";
}
```



- Try it to see what value **b** gets on your machine

3.18 A type-safety violation (Uninitialized variables)

```
// Beware: C++ does not prevent you from trying to use a variable
// before you have initialized it  (though a compiler typically warns)

int main()
{
    int x;              // x gets a "random" initial value
    char c;             // c gets a "random" initial value
    double d;           // d gets a "random" initial value
                        //      – not every bit pattern is a valid floating-point value
    double dd = d;      // potential error: some implementations
                                  // can't copy invalid floating-point values
    cout << " x: " << x << " c: " << c << " d: " << d << '\n';
}
```

- Always initialize your variables – beware: "debug mode" may initialize (valid exception to this rule: input variable)

3.19 A technical detail

- In memory, everything is just bits; type is what gives meaning to the bits

  (bits/binary) **01100001** is the int **97** is the char **'a'**
  (bits/binary) **01000001** is the int **65** is the char **'A'**
  (bits/binary) **00110000** is the int **48** is the char **'0'**

```
char c = 'a';
cout << c;    // print the value of character c, which is a
int i = c;
cout << i;    // print the integer value of the character c, which is 97
```

- This is just as in "the real world":
  - What does "42" mean?
  - You don't know until you know the unit used
    - Meters? Feet? Degrees Celsius? $s? a street number? Height in inches? …

3.20 About Efficiency

- For now, don't worry about "efficiency"
  - Concentrate on correctness and simplicity of code
- C++ is derived from C, which is a systems programming language
  - C++'s built-in types map directly to computer main memory
    - a **char** is stored in a byte
    - An **int** is stored in a word
    - A **double** fits in a floating-point register
  - C++'s built-in operations map directly to machine instructions
    - An integer + is implemented by an integer add operation
    - An integer = is implemented by a simple copy operation
  - C++ provides direct access to most of the facilities provided by modern hardware
- C++ help users build safer, more elegant, and efficient new types and operations using built-in types and operations.
  - E.g., **string**
  - Eventually, we'll show some of how that's done

3.21 A bit of philosophy

- One of the ways that programming resembles other kinds of engineering is that it involves tradeoffs.
- You must have ideals, but they often conflict, so you must decide what really matters for a given program.
  - Type safety
  - Run-time performance
  - Ability to run on a given platform
  - Ability to run on multiple platforms with same results
  - Compatibility with other code and systems
  - Ease of construction
  - Ease of maintenance
- Don´t skimp on correctness or testing
- By default, aim for type safety and portability

3.22 Another simple computation

```
// inch to cm and cm to inch conversion:

int main()
{
   const double cm_per_inch = 2.54;
   int val;
   char unit;
   while (cin >> val >> unit) { // keep reading
            if (unit == 'i')                    // 'i' for inch
                        cout << val << "in == " << val*cm_per_inch << "cm\n";
            else if (unit == 'c')   // 'c' for cm
                        cout << val << "cm == " << val/cm_per_inch << "in\n";
            else
                        return 0;  // terminate on a "bad unit", e.g. 'q'
   }
}
```

3.24 C++11 hint

- All language standards are updated occasionally
  - Often every 5 or 10 years
- The latest standard has the most and the nicest features
  - Currently C++14
- The latest standard is not 100% supported by all compilers
  - GCC (Linux) and Clang (Mac) are fine
  - Microsoft C++ is OK
  - Other implementations (many) vary

3.25 C++14 Hint

- You can use the type of an initializer as the type of a variable
  - // "auto" means "the type of the initializer"
  - auto x = 1; *// 1 is an **int**, so **x** is an **int***
  - auto y = 'c'; *// 'c' is a **char**, so **y** is a **char***
  - auto d = 1.2;*// 1.2 is a **double**, so **d** is a **double***
  - auto s = "Howdy"; *// "Howdy" is a string literal of type **const char[]***
    // so don't do that until you know what it means!
  - auto sq = sqrt(2); *// **sq** is the right type for the result of **sqrt(2)***
    // and you don't have to remember what that is
  - auto duh; *// error: no initializer for auto*

សំនួរ

១.តើ variable មានតួនាទីអ្វី?

២.តើvariable ក្នុង c++ មានលក្ខណៈជា case-sensitive ឬទេ?

៣.តើលក្ខខ័ណ្ឌបែបណាដែលមិនអាចដាក់ឈ្មោះឲ្យអញ្ញាត ឬ Identifier?

៤.ដើម្បីអានតម្លៃពីKeyboard ត្រូវប្រើ stream input អ្វី?ចូរឲ្យឧទាហរណ៍បញ្ជាក់។

The next lecture

- Will talk about expressions, statements, debugging, simple error handling, and simple rules for program construction