Basics: From C to C++

Computer Programming for Engineers (DASF003-41)

Instructor:

Sungjae Hwang

- jason.sungjae.hwang@gmail.com
- https://softsec-lab.github.io/

People

- Instructor
  - Sungjae Hwang (황성재, jason.sungjae.hwang@gmail.com)

- TAs
  - Bohyun Lee(이보현), lia323@skku.edu
  - Kyongshik Lee(이경식), kyongshikl@gmail.com

Calendar

Tentative Schedule

	Sun	Mon	Tue	Wed	Thu	Fri	Sat
W1	28	29	30	31	9/1	2	3
W2	4	5	6	7	8	9	10
W3	11	12(대체 휴일)	13	14	15	16	17
W4	18	19	20	21	22	23	24
W5	25	26	27	28 (PA1)	29	30	10/1
W6	2	3(개천절)	4	5	6	7	8
W7	9	10(대체 휴일)	11	12	13	14	15
W8	16	17	18	19(midterm)	20	21	22
W9	23	24	25	26 (PA2)	27	28	29
W10	30	31	11/1	2	3	4	5
W11	6	7	8	9	10	11	12
W12	13	14	15	16 (PA3)	17	18	19
W13	20	21	22	23	24	25	26
W14	27	28	29	30	12/1	2	3
W15	4	5	6	7(final)	8	9	10

Lab Sessions (20%)

- **Weekly assignments**
  - For checking your programming skill improvement.
  - There will be **12** WAs.

- **Getting points**
  - You should submit the weekly programming assignments (the problems you will discuss with the TAs).
  - Deadline: **12:00 PM on Fridays.**
  - If you miss the deadline, you do not have a chance to submit them.
  - If you submit only a subset of the problems or submit wrong answers, you cannot get a credit.

Intro.

- **C++ is a superset of C.**
  - This means you can try anything you can use in C.
  - All the basic types and control flow of C is accepted in C++.

- **C++ extends C in better/advanced ways.**
  - In this lecture, we will investigate many different/extended aspects of C++.
  - This material marks modern C++11, C++14, C++17… for some items; the others are typically a part of C++98 standard.

- **In case you are not familiar with C,**
  - Please intensively review the C programming immediately.
  - In this course, I will always presume you are familiar with C programming.

Goal

Textbook : Absolute C++ 6th edition (Walter Savitch)

- Variables & Types (1.2, 1.4)
- Variable Declaration and Type Deduction (1.2)
- Constant & Enumeration (1.2, 2.2)
- Program Style (1.4)
- Namespace (1.5)
- Console I/O (Briefly) (1.3)

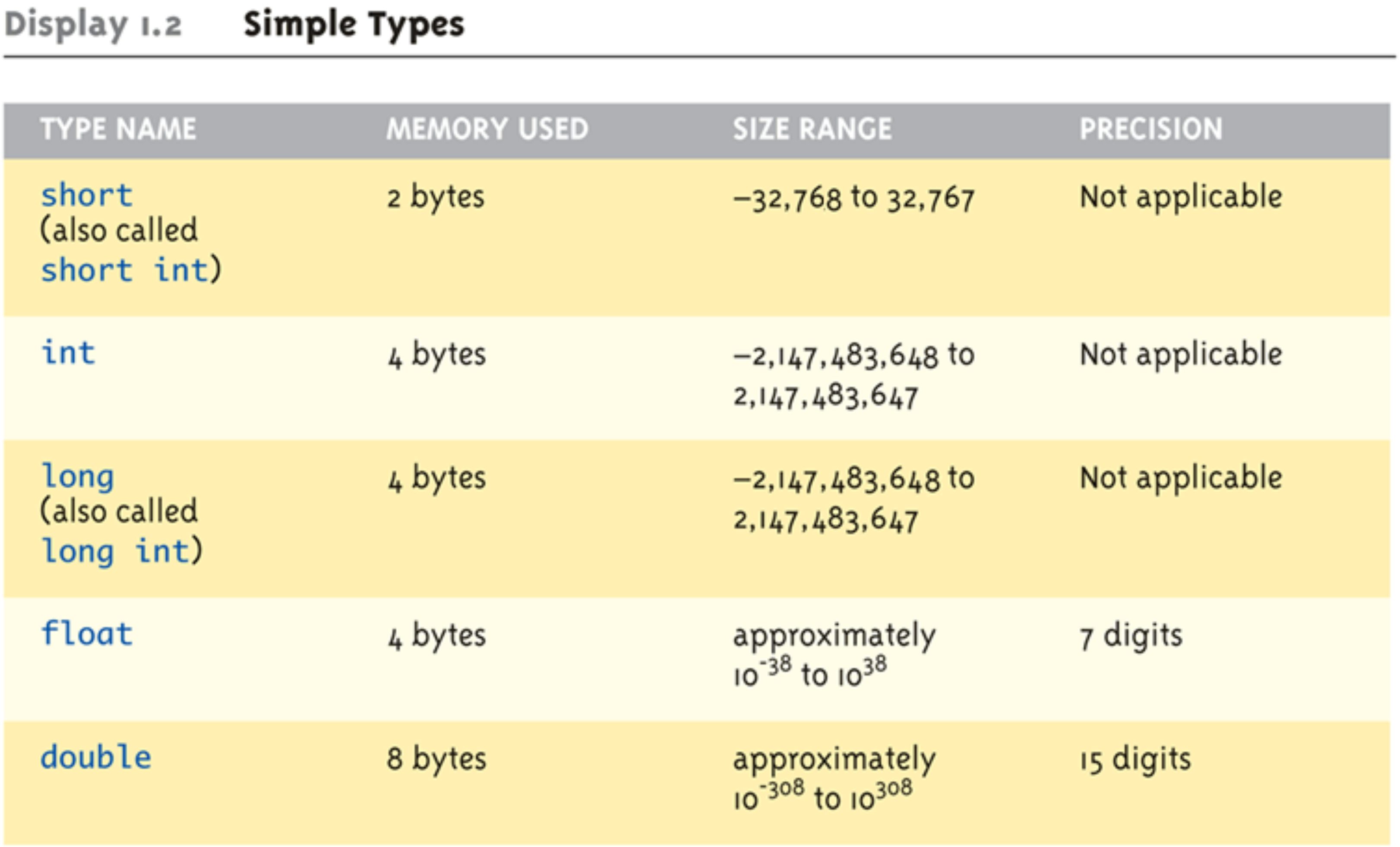
VARIABLES & TYPES

C++ Variables

- **Variables**
  - A memory location to store data for a program
  - Must declare all data before use in program

- **C++ Identifiers**
  - Keywords/reserved words vs. Identifiers
    - if, new, char, int, do, while
  - Case-sensitivity and validity of identifiers
    - rate, RATE, Rate
    - x, x1, _a
    - 12, 3X, %change
  - Meaningful names
    - x=11 vs studentAge=11

Tyes in C/C++

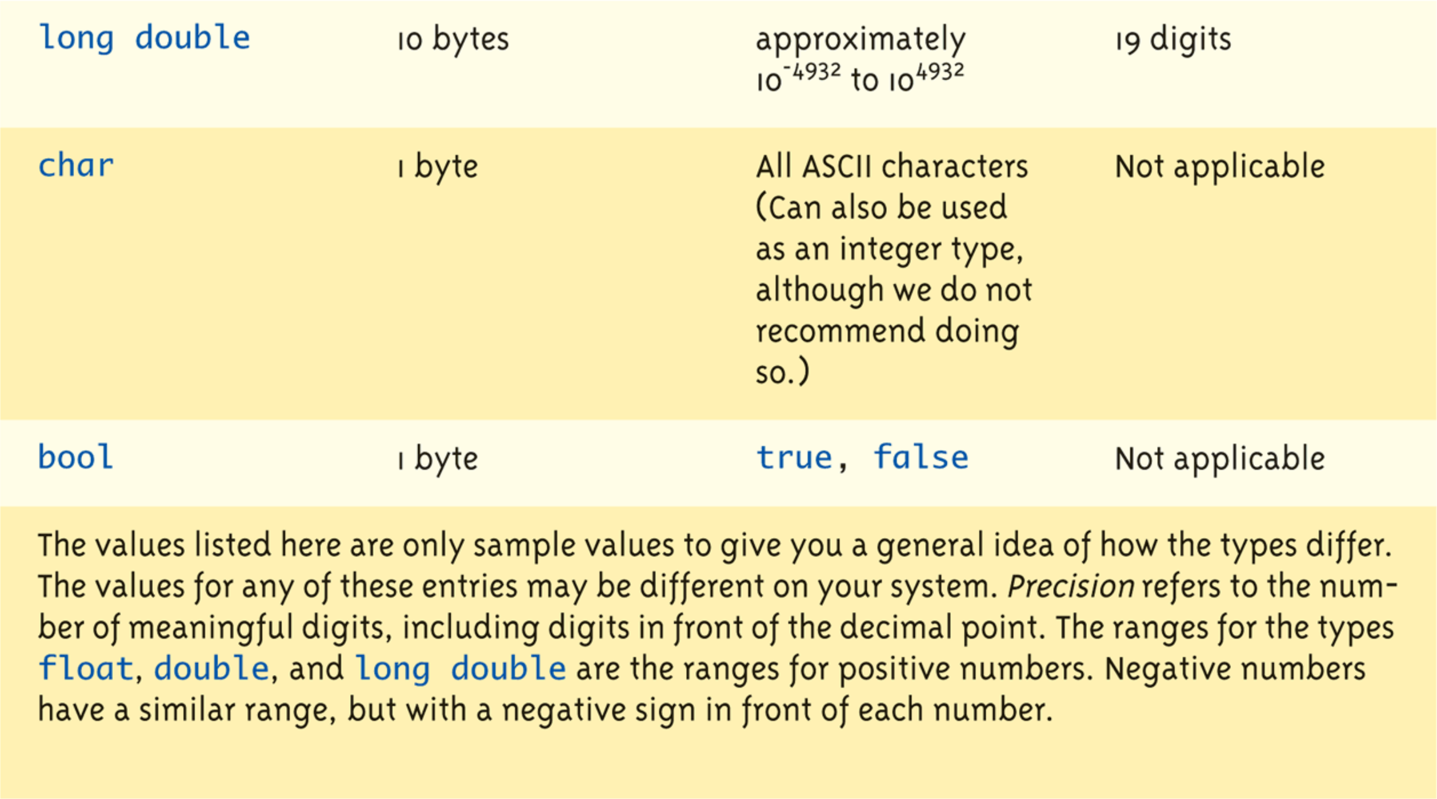


Display 1.2 **Simple Types**

TYPE NAME	MEMORY USED	SIZE RANGE	PRECISION
short (also called short int)	2 bytes	−32,768 to 32,767	Not applicable
int	4 bytes	−2,147,483,648 to 2,147,483,647	Not applicable
long (also called long int)	4 bytes	−2,147,483,648 to 2,147,483,647	Not applicable
float	4 bytes	approximately 10^{-38} to 10^{38}	7 digits
double	8 bytes	approximately 10^{-308} to 10^{308}	15 digits

Tyes in C/C++

- **Note:**
  - Bool is not a standard C type, but is in C++
  - long double is equivalent to double in VC++:i.e., 8 bytes

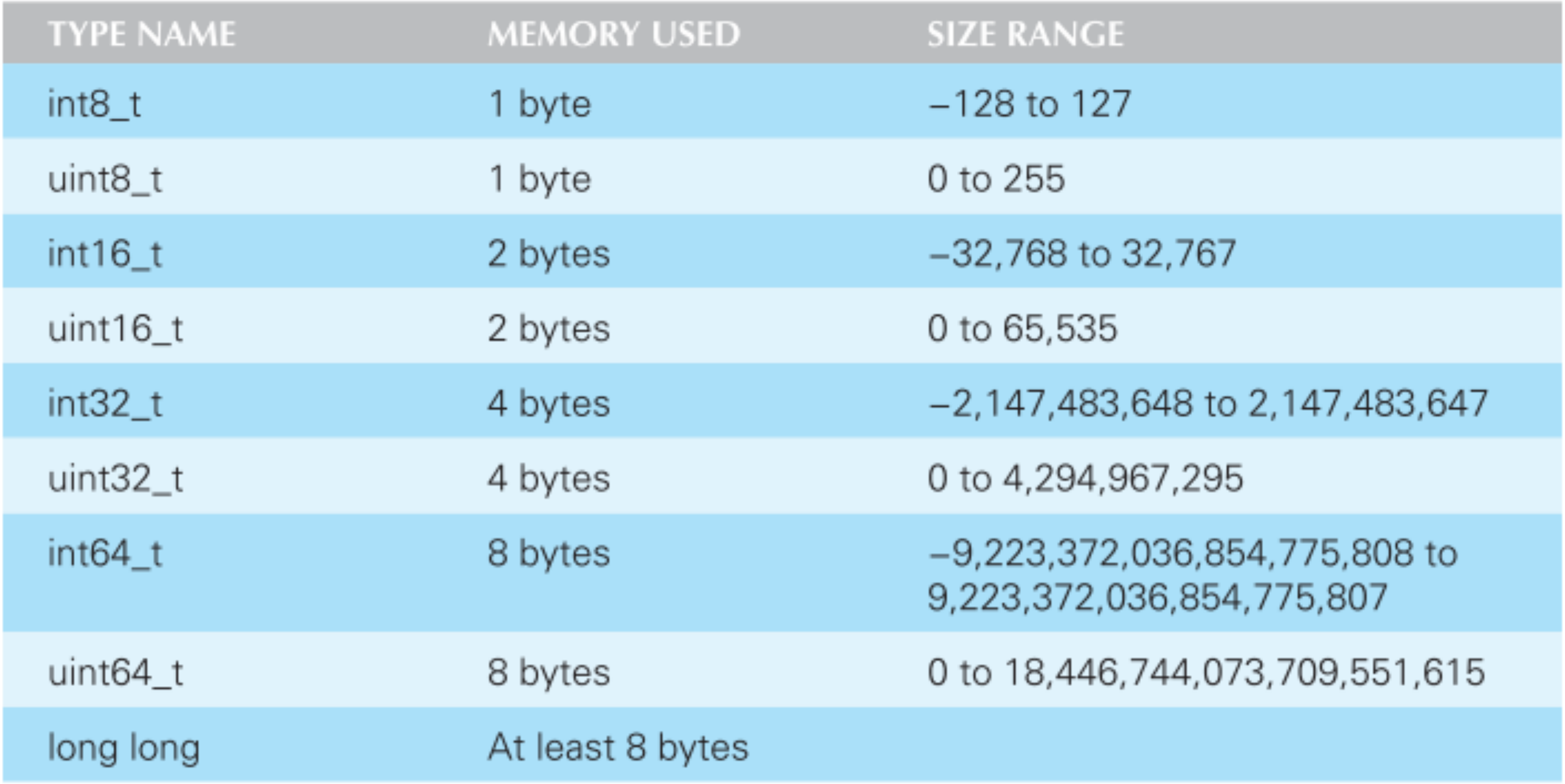


`long double`	10 bytes	approximately 10^{-4932} to 10^{4932}	19 digits
`char`	1 byte	All ASCII characters (Can also be used as an integer type, although we do not recommend doing so.)	Not applicable
`bool`	1 byte	`true, false`	Not applicable

The values listed here are only sample values to give you a general idea of how the types differ. The values for any of these entries may be different on your system. *Precision* refers to the number of meaningful digits, including digits in front of the decimal point. The ranges for the types `float`, `double`, and `long double` are the ranges for positive numbers. Negative numbers have a similar range, but with a negative sign in front of each number.

New Integral Types (C++11)

- Avoids problem of variable integer size for different CPUs



TYPE NAME	MEMORY USED	SIZE RANGE
int8_t	1 byte	–128 to 127
uint8_t	1 byte	0 to 255
int16_t	2 bytes	–32,768 to 32,767
uint16_t	2 bytes	0 to 65,535
int32_t	4 bytes	–2,147,483,648 to 2,147,483,647
uint32_t	4 bytes	0 to 4,294,967,295
int64_t	8 bytes	–9,223,372,036,854,775,808 to 9,223,372,036,854,775,807
uint64_t	8 bytes	0 to 18,446,744,073,709,551,615
long long	At least 8 bytes	

size_t (since C98)

- **An unsigned data type defined by C/C++ standards.**
  - typically, it represents the size of types/variables in terms of bytes

```
size_t int_size = sizeof(int);
size_t double_size = sizeof(double);

printf( "%zu %zu\n", int_size, double_size );
```

```
>> 4  8
```

sizeof example

DEMO

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

 int main(){
   size_t int_size = sizeof(int);
   size_t u_int_size = sizeof(unsigned int);
   size_t short_size = sizeof(short);
   size_t u_short_size = sizeof(unsigned short);

   // use 'z' for a length specifier in C
   // printf("%zd %zd %zd\n", int_size, double_size, float_size);
   cout << "* int size: " << int_size << endl;
   cout << "* (unsigned) int size: " << u_int_size << endl;
   cout << "* short size: " << short_size << endl;
   cout << "* (unsigend) short size: " << u_short_size << endl;
   return 0;
 }
```

Raw String Literals (C++11)

- Newly introduced with C++11
- Avoids escape sequences by literally interpreting everything in parentheses

```
string s = R"(\t\\t\n)";
```

  - The variable s is set to the exact string “`\t\\t\n`”
  - Without "R", s is interpreted as tap, backslash, 't', newline.

- Useful for filenames with \ in the file path

String example

DEMO

```
#include <iostream>

using namespace std;

int main()
{
  string s = "(\t\\t\n)";
  //string s = R"(\t\\t\n)";

  cout << s << endl;

  return 0;
}
```

VARIABLE DECLARATION AND TYPE DEDUCTION

Variable Declaration Anywhere

- In C, you needed to pre-declare all the variables before you use in your functions. (C99 allows it.)
  - C++ relaxes this constraint significantly.
  - You can declare variables (nearly) anywhere, as long as syntax allows.
- Example: for loop
  - C style

```
int k;
...
for( k=0; k < 10; k++ ) printf( “%d\n”, k );
```

  - C++ style

```
for( int k=0; k < 10; k++ ) printf( “%d\n”, k );
```

Automatic Type Deduction (C++11)

- **new ‘auto’ keyword**
  - auto in C (meaning local variable) deprecated
  - Deduces the type of the variable based on the expression on the right side of the assignment statement

```
auto x = expression;
```

  - More useful later when we have verbose types
    - e.g., iterators in STL (standard template library)

```
std::unordererd_map<int,std::string>::iterator it = m.begin();
auto it2 = m.begin(); // type deduced automatically
```

auto example

DEMO

```
#include <iostream>

using namespace std;

int main()
{
  auto x = 10;
  auto y = "Ten";
  cout << x << " is " <<  y << endl;
}
```

Automatic Type Deduction (C++11)

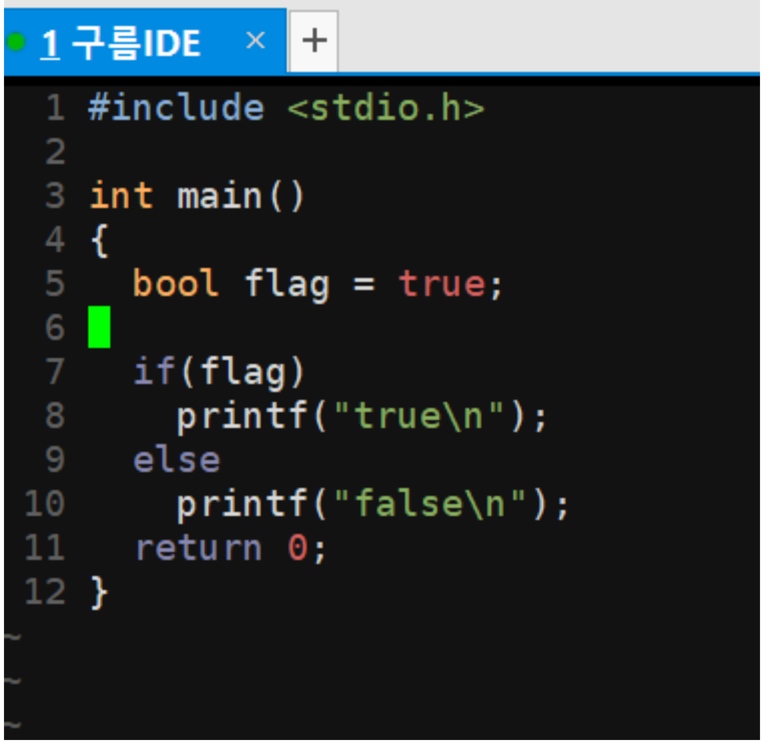
- **‘decltype’**
  - automatic type deduction by the existing variable or expression
  - Determines the type of the expression. In the example below, x*3.5 is a double so y is declared as a double.

```cpp
double x=1.0;
decltype(x*3.5) y=2.0;
```

Short Demo

What happens with the following code

- when compiling with C compiler?
- when compiling with C++ compiler?



```
#include <stdio.h>

int main()
{
  bool flag = true;

  if(flag)
    printf("true\n");
  else
    printf("false\n");
  return 0;
}
```

CONSTANTS AND ENUMERATION

Named Constants in C++

- **Naming your constants**
  - Literal constants (e.g., 24) are "OK", but provide little meaning
    - e.g., 24 tells nothing about what it represents.

- **Use named constants instead**
  - Meaningful name to represent data

```cpp
const int NUM_STUDENTS = 24;
```

    - Called a “declared constant” or “named constant”.
    - Now use it’s name wherever needed in program.
    - Added benefit: change to value result in one fix.

Named Constants in C++

- Named constant

```
Enter the amount of your deposit $100
In one year, that deposit will grow to
$106.9 an amount worth waiting for.
```

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

int main( )
{
   const double RATE = 6.9;
   double deposit;
   cout << "Enter the amount of your deposit $";
   cin >> deposit;
   double newBalance;
   newBalance = deposit + deposit*(RATE/100);

   cout << "In one year, that deposit will grow to\n"
   << "$" << newBalance << " an amount worth waiting for.\n";
   return 0;
}
```

Enum

- enum can be used to systematically declare multiple (having sequential values) constants.

```
enum Direction { NORTH, SOUTH, EAST, WEST };
enum Direction { NORTH=0, SOUTH=1, EAST=2, WEST=3 };
```

  - each item type in enum is assumed to be an integer.

```
enum MODE { WEAPON, EQUIPMENT, GEM = 10, DEFENSE, };
```

- Handy for switch statement

enum

```
#include <iostream>
using namespace std;
int main() {
  enum MODE { WEAPON, EQUIPMENT, GEM,  DEFENSE};
  int mode;
  cout << "Enter mode(0:Weapon, 1:Equipment, 2:Gem, :Defence): ";
  cin >> mode;
  switch(mode) {
  case WEAPON:
    cout << "Weapon" << endl; break;
  case EQUIPMENT:
    cout << "Equipment" << endl; break;
  case GEM:
    cout << "Gem" << endl; break;
  case DEFENSE:
    cout << "Defence" << endl; break;
  case default:
    cout << "Wrong mode" << endl;
  }
}
```

Strong enum (C++11)

- **C++11 introduces strong enums or enum classes**
  - Does not act like an integer.
  - Solved scope problem
  - Examples

```
enum class Days { Sun, Mon, Tue, Wed, Thu, Fri, Sat };
enum class Weather { Rain, Sun };

Days d = Days::Tue;
Weather w = Weather::Sun;
```

- Illegal: if (d == 0)
- Legal: if (d == Days::Wed)

Strong enum

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

int main() {
  //scope problem
  //enum x { Error, Ok};
  //enum y { Error, Ok};

  enum class IOResult {Error, Ok};
  enum class ParseResult {Error, Ok};
  IOResult io_return_code = IOResult::Ok;


  switch(io_return_code) {
    //case 1:
    case IOResult::Ok:
      cout << "IO done" << endl; break;
    case IOResult::Error:
      cout << "IO Error" << endl;
  }
 }
```

COMMENTS AND STYLES

Comments

- Two methods:

```
// Two slashes indicate entire line is to be ignored
/* Delimiters indicates
   everything between is ignored */
```

  - Both methods commonly used
  - Note: /// is the same as //, but often used as a meta information indicator.
    - The comment following /// is used for automatic document generation (e.g., doxygen)

Program Style

- **Remember the following basic rule:**
  - Make programs easy to read, modify, and maintain!

- **Naming conventions of identifiers (Book's author)**
  - constants: ALL_CAPS
  - variables: lowerToUpper or under_bar_var
  - Most important: make MEANINGFUL NAMES!
    - In case the variable name is meaningful, you do not have to add much comments to explain what the variables are for.

NAMESPACES AND LIBRARIES

Namespaces

- **Namespaces defined:**
  - Collection of name definitions (class definition, variable declarations)
  - For now: interested in namespace “std”
  - Has all standard library definitions we need\
- **Examples:**

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

  - Includes entire standard library of name definitions.

```
#include <iostream>
using std::cin;
using std::cout;
```

  - Can specify just the objects we want.

Libraries

- C++ Standard Libraries
- #include <Library_Name>
  - Directive to "add" contents of library file to your program
  - Called "preprocessor directive"
    - Executes before compiler, and simply "copies“ library file into your program file

- C++ has many libraries
  - Input/output, math, strings, etc.

CONSOLE I/O

Console I/O

- **I/O objects cin, cout, cerr**
  - Defined in the C++ library called <iostream>
  - Must have these lines (called preprocessor directives) near start of file:

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

  - Tells C++ to use library so we can use the I/O objects cin, cout, cerr

Input Using cin

- **cin for input, cout for output**
- **Differences:**
  - “>>” (extraction operator) points opposite
    - Think of it as “pointing toward where the data goes"
  - Object name “cin” used instead of “cout”
  - No literals allowed for cin
    - Must input “to a variable”

- **cin >> num;**
  - Waits on-screen for keyboard entry
  - Value entered at keyboard is “assigned” to num

Input/Output Example

- Using cin and cout

```
//Program to demonstrate cin and cout
#include <iostream>
using namespace std;

int main( )
{
    cout << "Enter your number: “;

    int num;
    // The variable num will hold a value
    // from a keyboard as a user input
    cin >> num;
    cout << "Entered number: " << num << endl;
    return 0;
}
```