Lecture #01

Variables, Type Conversion, Basic Operators, Comments

SE271 Object-Oriented Programming (2020)

Yeseong Kim

Original slides from Prof. Shin at DGIST

Short Notice

- Posted in LMS
  - First lecture note
  - Second lecture note (this one)
    - Will update it after the class if changed
  - Zoom link for office hours

- TODO: Unofficial assignment
  - Install your IDE and test example programs

Today's Topic

- Variables
  - Naming
  - Type
  - Initialization
- Type Conversion
- Basic Operators
  - Precedence

예제: Hello World!

```
/* 예제 파일 */
#include <iostream>
using namespace std;
int main()
{
    int num1, num2;
    num1 = 1; num2 = 2;
    // print a string 'hello world!'
    std::cout << "Hello, World! " << endl;
    cout <<num1<< "+" <<num2 << "=" <<num1+num2;
    return 0;
}
```

Comments
Header files
namespace
function : main()
Statement
Keywords
Identifiers
Literals
Code block
Standard output
Punctuation
Operators
Whitespace

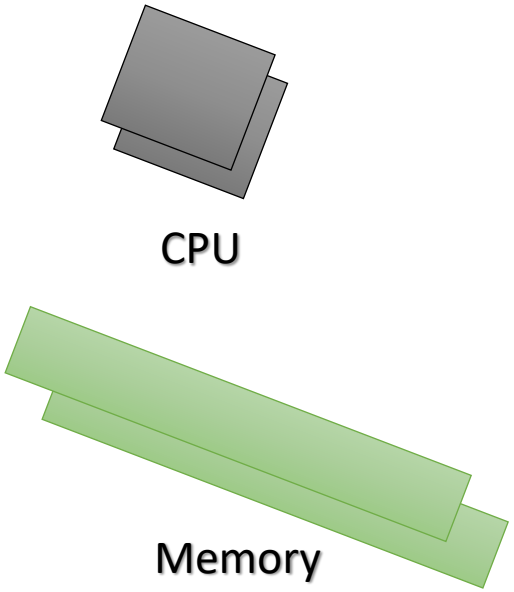
```
Hello, World!
1+2=3
```

Programming Language

- Doing something
  - (computer) some data – some operation(process)

- Essential computer components

  CPU – Memory



ALU – Register – Memory – HDD(SSD)

Variable

- Storage location for the **data**

- Variables
  - **Declaration**

Usage:	`type variable_name;`

    - Name(identifier):
      1. Consists of letters(case sensitive), numbers, '_'
      2. Starts with letters or '_'
      3. Allows no space

    - **Type** ← cf. Python
    - Initialization

Variable Type

- Basic Type
  - Integer type
  - Floating point type
  - Character type
  - Boolean type

std::int16_t

Integer	Memory size	Data
(signed) **short** (or short int)	2 Bytes	$-2^{15} \sim (2^{15} - 1)$
unsigned short	2 Bytes	$0 \sim (2^{16} - 1)$
(signed) **int**	4 Bytes (2)	$-2^{31} \sim (2^{31} - 1)$
unsigned int	4 Bytes	$0 \sim (2^{32} - 1)$
(signed) **long**	4 Bytes	$-2^{31} \sim (2^{31} - 1)$
unsigned long	4 Bytes	$0 \sim (2^{32} - 1)$

unsigned

1 Byte ?

C++11

18,446,744,073,709,551,616

Variable Type

- Basic Type
  - Integer type
  - **Floating point type**
  - Character type
  - Boolean type

Floating point	Memory size	Data
float	4 Bytes	$(3.4 \times 10^{-38}) \sim (3.4 \times 10^{38})$
double	8 Bytes	$(1.7 \times 10^{-308}) \sim (1.7 \times 10^{308})$
long double	8? Bytes	8/12/16 depends on system

Q1. How to represent?

Q2. Is it OK?

Variable Type

- Basic Type
  - Integer type
  - Floating point type
  - **Character type**
  - Boolean type (bool)
    - C/C++ : 0 == false, non-zero==true
    - cf. C++11: false/true

Character Type	Memory size
char	1 Byte
wchar_t	2 Bytes
char16_t (C++11)	2 Bytes
char32_t (C++11)	4 Bytes

Unicode decoding : utf-8, utf-16…

Q1. ‘c’ == u‘c’ ?

Variable Size

- sizeof() : return the memory size

```
cout << "char type: " << sizeof(char) << " Bytes" << endl;
cout << "int type: " << sizeof(int) << " Bytes" << endl;
cout << "short type: " << sizeof(short) << " Bytes" << endl;
cout << "long type: " << sizeof(long) << " Bytes" << endl;
cout << "long long type: " << sizeof(long long) << " Bytes" << endl;
cout << "float type: " << sizeof(float) << " Bytes" << endl;
cout << "double type: " << sizeof(double) << " Bytes" << endl;
```

```
char type: 1 Bytes
4
2
4
8
4
8
```

Extended Variable Type (typeid())

- **auto** type (C++11)
  - At the initialization, the actual type is decided. (type inference)

```
int iValue = 20;
auto aValue1 = iValue;
auto aValue2 = 10.5;
std::cout << typeid(aValue2).name();
```

double

- String?
  - “a” vs. ‘a’
  - char sName[] = “John”;
  - std::string sName = “John”;
  - cf. wchar_t sName[] = L“John”;
  - cf. char16_t sName[] = u“John”;

Q1. sizeof(sName) ?

Variable initialization

- Initial assignment with declaration
  - Copy initialization
  - Direct initialization
  - Uniform initialization
    - It prevents improper conversions

```
int iNum1 = 10.5;
int iNum2(30.5);
int iNum3{ 20.5 }; // Error
```

```
int iNum1 = 10;
int iNum2(30);
int iNum3{ 20 }; // c++11 {}
```

Type Conversion

- Implicit type conversion
  * (to minimize information lose)
  – Assignment
  – Arithmetic expressions
    • Match to the higher type of operands
    • int → long → float → double

- Explicit type conversion
  – (type) data // c style
  – type(data) // c++ style
  – static_cast<type>(data) // c++11 style
    • cf. const_cast, dynamic_cast, reinterpret_cast

```
int iVal = 3.1;
cout << iVal << endl;
double dVal = 3.1f;
cout << dVal << endl;
long lVal;
lVal = iVal;
cout << lVal << endl;
```

```
cout << (double)(2 / 4) << endl;
cout << ((double)2 / 4) << endl;
cout << (double(2) / 4) << endl;
cout << (static_cast<double>(2)/4);
```

Constant

- Fixed values that the program may not alter
  - 10, 31.4, 3.14f, “test”, ‘c’, …

- C-Style
  - #define PI 3.14

- C++ Style
  - **const** float PI = 3.14; // constexpr

```
const double gravity{ 9.8 };
gravity = 10; // Error
```

Basic Operators(1)

- Arithmetic operator(binary operators, left ←right)

  +, -, *, /, %

```
int iVal = 3;
double dVal = 3.1;
dVal = 2 + iVal;
cout << dVal << endl;
```

- Assignment operator(binary operators, left ←right)

  =, +=, -=, *=, /=, %=

```
int iVal = 3;
double dVal = 3.1;
dVal += iVal;
cout << dVal << endl;
```

- Increment and decrement operator(unar

  ++, -- (Prefix, Posfix)

```
int iVal1 = 3;
int iVal2 = iVal1++;
cout << iVal1 << "₩t" << iVal2<< endl;

iVal1 = 3;
iVal2 = ++iVal1;
cout << iVal1 << "₩t" << iVal2 << endl;
```

Basic Operators(2)

- **Comparison operators**

  ==, !=, >, >=, <, <=

- **Logical operators**

  &&, || (binary operator), ! (unary operator)

- **Bitwise operators**

  &, |, ^, ~, <<, >>

Q1. iVal = 1 + 2 + 3;

Q2. iVal = 1 + 2 * 3;

Q3. iVal = iVal2 = 3;

```
int iVal = 1;
cout << iVal << (iVal << 1);
cout << (iVal << 2) << (iVal << 3);
```

Operator Precedence and Associativity

- **Operator precedence**
  - **ex) 1 + 2 * 3 ➔ (1 + (2 * 3))**

- **Operator associativity**
  - **ex) 1 + 2 + 3 ➔ ((1 + 2) + 3)**

Operator Precedence

- (L->R) Type parentheses, post-increment/decrement, Type cast
- (R->L) sign, c-style cast, pre-increment/decrement
- (L->R) *, /, %
- (L->R) +, -
- (L->R) <<, >>
- (L->R) <, <=, >, >=
- (L->R) ==, !=
- (L->R) bit operators(&)
- …
- (L->R) logical operators(&&)
- (R->L) assignment

Use ( ) if necessary!

```
int x = 1, y = 2;
cout << (++x > 2 && ++y > 2);
cout << "x=" << x << ", y=" << y << endl;
cout << (++x > 2 || ++y > 2);
cout << "x=" << x << ", y=" << y << endl;
```

https://www.learncpp.com/cpp-tutorial/operator-precedence-and-associativity/

References

- Learn c++
  - https://www.learncpp.com/
  - Chapter 1, 4
  - Chapter 5.1, 5. 4

예제: Circumference Calculator

```
#include <iostream>
#define PI 3.14
using namespace std;
int main()
{
    int iValue(2);
    double circum{0};
    for (;;) {
        cout << "Enter the radius? ";
        cin >> iValue;
        if (iValue == 0)
          break;    circum = iValue * 2 * PI;
        cout << "circumference with a radius of " << iValue << " : ";
        cout << circum << "(" << typeid(circum).name() << ")" << endl;
    }
    return 0;
}
```

ANY QUESTIONS?