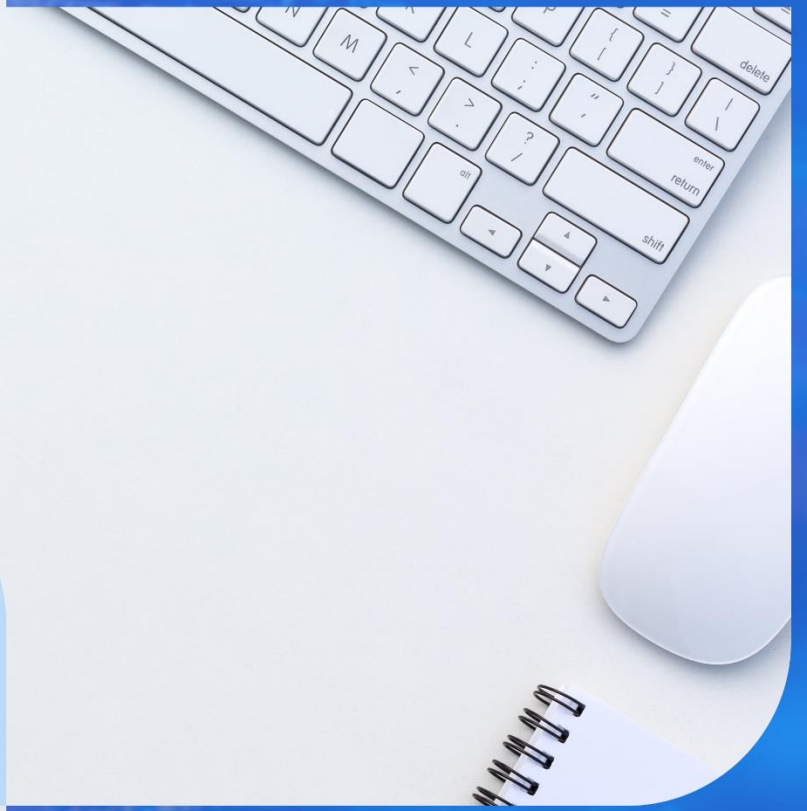
Basis and Practice in Programming

SUNG KYUN KWAN UNIVERSITY(SKKU)

Programming Basics 1



Instructor Younghoon Kim

Basic Concepts

Syntax and Semantics

- **Syntax**
  - The rules of C programming language
  - The set of rules that defines the combinations of symbols that are correctly structured statements or expressions

- **Semantics**
  - meaning associated with each syntactically correct sequence of characters.

- **Syntax and Semantic Errors**
  - E1: “what tyme is it now?”
  - E2: “A silent noise drinks stones.”

Comments

- Additional information that is not part of a program
- Syntax
  - // Just for one line
  - /* Multiple number of lines */

Preprocessor Directives

- For telling the preprocessor to take specific actions
- Syntax
  - #
- Example
  - #include, #ifdef, #define

Block Structures

- Block: one or more declarations and statements
- Nested blocks are permitted.
- Syntax
  - » { <lines of code> }

Statement

- A command instructing a computer to take a specific action
- Syntax
  - » <a line of code>;

Expression

- An expression is a collection of operators and operands.
- An expression can be reduced to some kind of "value".
- Expressions are parts of statements.
- Typical example:

```
int sum = num1 + num2 + num3;
```

» `'num1 + num2 + num3'` is an expression.

» The whole line is a statement.

Hello! World! (Again)

SUNG KYUN KWAN UNIVERSITY(SKKU)

- Let's check the explained C syntax with an example.

```
// This program prints a string,
// "Hello! World!" on the screen.

#include <stdio.h>
int main()
{
        printf("Hello! World!\n");
        return 0;
}
```

Variables

Variable

- **A program temporarily stores data in variables.**
  - All variables must be defined before they can be used in a program.
- **Syntax for declaring a variable**
  - <type> <variable_name>
- **Data types**
  - `char`, `int`, `float`, `double` , …
  - pointers

```
char name[20];          //name: name, type: char array
int score;              //name: score, type: int
```

Variable (cont.)

- **Naming Rules**
  - Variable names are identifiers in C.
  - Letters, digits and underscores are permitted.
  - Variable names cannot start with digits.
  - **Case sensitive**: Uppercase and lowercase letters are different

- **Assignment**
  - Assigning values to variables
  - Assigned values are used until re-assigned by the program.

Example of Naming Variables

DEMO

below code has an error

```c
#include <stdio.h>

int main()
{
  int num1;
  int _Num2;
  int nuM1;
  int 1sum; // Wrong name!

  // Initializing section
  num1 = 1;
  _Num2 = 2;
  nuM1 = 3;
  1sum = num1 + _Num2 + num3;  // Unknown variable

  printf("sum1: %d\n", 1sum);

  return 0;
}
```

Variable Declarations

Variable declarations with Initializations

- We can give initial values for variables while declaring them.
- Simply done by using assignment operators.

```
int num1 = 1;
```

Multiple variable declarations

- Variables with the same data types can be declared using comma operators.

```
int num1, num2, num3;
```

Mixture of both is also possible

```
int num1 = 1, num2 = 2, num3 = 3;
```

Variable Declarations

DEMO

- Previous example code can be written much simpler.

```c
#include <stdio.h>

int main()
{
  int num1 = 1, num2 = 2, num3 = 3;
  int sum = num1 + num2 + num3;
  printf("sum1: %d\n", sum);
  return 0;
}
```

Example of Naming Variables

DEMO

below code has an error

```c
#include <stdio.h>

int main()
{
  int num1;
  int _Num2;
  int nuM1;
  int 1sum; // Wrong name!

  // Initializing section
  num1 = 1;
  _Num2 = 2;
  nuM1 = 3;
  1sum = num1 + _Num2 + num3; // Unknown variable

  printf("sum1: %d\n", 1sum);

  return 0;
}
```

Identifiers and Case Sensitivity

- A variable name in C is any valid identifier. An identifier is a series of characters consisting of letters, digits and underscores (_) that does not begin with a digit.
- C is case sensitive—uppercase and lowercase letters are different in C, so a1 and A1 are different identifiers.

Assignment Statement:

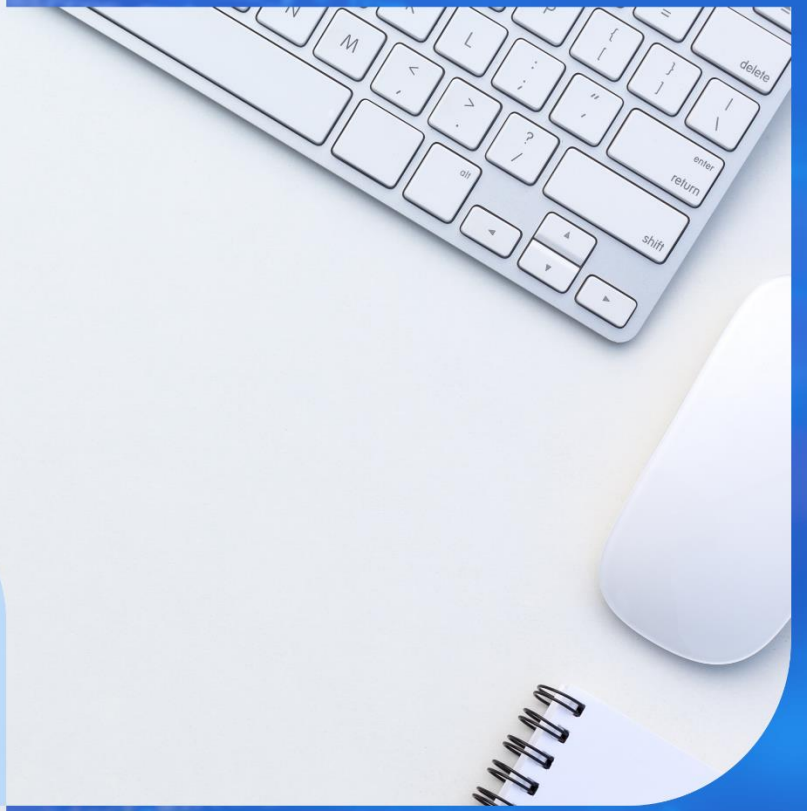
```
sum = integer1 + integer2; // assign total to sum
```

- calculates the total of variables integer1 and integer2 and assigns the result to variable sum using the assignment operator =.
- The statement is read as, “sum gets the value of integer1 + integer2.”
- The = operator and the + operator are called binary operators because each has two operands.
- The + operator’s two operands are integer1 and integer2.
- The = operator’s two operands are sum and the value of the expression integer1 + integer2.

Basis and Practice in Programming

SUNG KYUN KWAN UNIVERSITY(SKKU)

Programming Basics 1



Instructor Younghoon Kim

Standard Input and Output

#include <stdio.h>

SUNG KYUN KWAN UNIVERSITY(SKKU)

- **stdio.h**
  - A ***header file*** in which ***functions*** for standard input/output are ***defined.***

- **#include < *a_file* > or #include " *a_file* "**
  - One of preprocessor directives
  - It imports source code from *a_file*.

- **#include <stdio.h>**
  - You should write this statement in your source code to utilize standard input/output functions.

- **formatted output (print formatted)**
  - You can give a format in which you want to see from the console.
  - Ordinary or special characters and conversion specifications

- **Syntax**
  - printf(***format***, . . .);
  - ***format*** : " *characters and/or conversion specifications* "

- **Example without conversion specifications**

```
printf("I love python.");
```

```
I love python.
```

Special Characters

- **You can use following special characters.**

Special Character	Escape Sequence
newline	\n
tab	\t
backslash	\\
double quote	\"

- There can be more escape sequences for more special characters.

- **Example with special characters**

```
printf(" I love \"python\".\n");
```

```
 I love "python".
>> (newline)
```

Conversion Specifications

SUNG KYUN KWAN UNIVERSITY(SKKU)

- **You can dynamically manipulate the printing result.**
- **Conversion specifications in the *format* are substituted with following arguments.**

```
int age = 100;
printf("Age: %d\n", age);
```

```
Age: 100
>> (newline)
```

- In the 2nd line, %d is substituted with the value of age.

- **Types of conversion specifications**
  - `%d` : integer, `%f` : float, `%lf` : double, `%c` : character, `%s` : string, …
  - **Compilers may not check inappropriate conversion specifications.**

- **formatted input (scan formatted)**
  - You can give a format in which you want to receive from the user.
  - Ordinary or special characters and conversion specifications
  - For now, it is recommended to use only conversion specifications for `scanf`.

- **Syntax**
  - scanf(***format***, . . .);
  - ***format*** : " *characters and/or conversion specifications* "

- **Use proper conversion specifications.**

```
int age;
scanf("%d", &age);          // &: 'address-of' operator
                            // mandatory for normal variables
```

Variable Declarations (Re-visited)

DEMO

- **Previous example code can be written much simpler.**

```c
#include <stdio.h>

int main()
{
  int num1 = 1, num2 = 2, num3 = 3;
  int sum = num1 + num2 + num3;
  printf("sum1: %d\n", sum);
  return 0;
}
```

- **Follow-up question**
  - Print multiple variable with one `printf`.

Example of Naming Variables DEMO

below code has an error

```c
#include <stdio.h>

int main()
{
  int num1;
  int _Num2;
  int nuM1;
  int 1sum; // Wrong name!

  // Initializing section
  num1 = 1;
  _Num2 = 2;
  nuM1 = 3;
  1sum = num1 + _Num2 + num3; // Unknown variable

  printf("sum1: %d\n", 1sum);

  return 0;
}
```

DEMO

- **Execute and do some variations on it.**

```
#include <stdio.h>

int main()
{
  int current_age, duration;
  printf("Enter your age : ");
  scanf("%d", &current_age);
  printf("Your age is %d,\n", current_age);
  printf("and now you fall asleep…\n\n\n");

  printf("Enter sleep duration : ");
  scanf("%d", &duration);
  printf("You were %d years old, but now, %d.\n", current_age, current_age + duration);
  return 0;
}
```

The scanf Function and Formatted Inputs

```
scanf( "%d", &integer1 ); // read an integer
```

- uses scanf to obtain a value from the user. The scanf function reads from the standard input, which is usually the keyboard.
- scanf has two arguments, "%d" and &integer1.
- The first, the format control string, indicates the type of data that should be input by the user. The %d conversion specifier indicates that the data should be an integer. The % in this context is treated by scanf (and printf) as a special character that begins a conversion specifier.
- The second argument of scanf begins with an ampersand (&)—called the address operator in C—followed by the variable name. The &, when combined with the variable name, tells scanf the location (or address) in memory at which the variable integer1 is stored. The computer then stores the value that the user enters for integer1 at that location.
- When the computer executes the preceding scanf, it waits for the user to enter a value for variable integer1. The user responds by typing an integer, then pressing the Enter key to send the number to the computer. The computer then assigns this number, or value, to the variable integer1. Any subsequent references to integer1 in this program will use this same value. Functions printf and scanf facilitate interaction between the user and the computer.17

Printing with a Format Control String

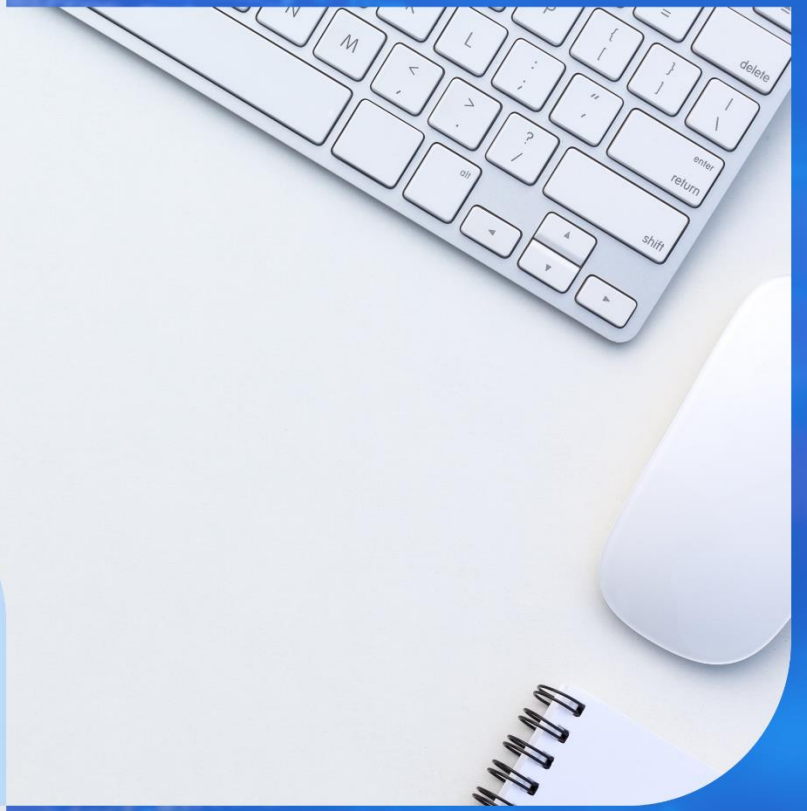
```
printf( "Sum is %d\n", sum ); // print sum
```

- calls function printf to print the literal Sum is followed by the numerical value of variable sum on the screen.
- This printf has two arguments, "Sum is %d\n" and sum.
- The first argument is the format control string. It contains some literal characters to be displayed, and it contains the conversion specifier %d indicating that an integer will be printed.
- The second argument specifies the value to be printed. Notice that the conversion specifier for an integer is the same in both printf and scanf.

Basis and Practice in Programming

SUNG KYUN KWAN UNIVERSITY(SKKU)

Programming Basics 1



Instructor Younghoon Kim

Operators

Binary Arithmetic Operators

Operation	Operator	C expression
Addition	+	a + b
Subtraction	-	a – b
Multiplication	*	a * b
Division	/	a / b
Remainder	%	a % b

Assignment Operator

- v = e is to evaluate the expression e and copy its value into v

```
i = 5;                          /* i is now 5  */
j = i;                          /* j is now 5  */
k = 10 * i + j;                 /* k is now 55 */
```

Integer division and Remainder OP

SUNG KYUN KWAN UNIVERSITY(SKKU)

- **Integer division yields an integer result**
  - The results are the quotients of division. (Floor operation is used implicitly.)
  - Ex: 7 / 4 → 1, 17 / 5 →3

- **Remainder operator (%)**
  - yields the remainder after integer division
  - can be used only with integer operands
  - Ex: 7 % 4 → 3, 17 % 5 → 2

Remainder OP with Negative Numbers

SUNG KYUN KWAN UNIVERSITY(SKKU)

- **Remainder operator with negative numbers**
  - Follows the arithmetic. (divisor x quotient + remainder = dividend)

First Operand	Second Operand	Division	Remainder
7	4	1	3
7	-4	-1	3
-7	4	-1	-3
-7	-4	1	-3

Other Operators

- Compound assignment operator

```
i = 5;                  /* i is now 5  */
i = i + 5;              /* i is now 10  */
i += 5;                 /* i is now 15 */
```

» -=, *=, /=, %=

- Increment and decrement operators

```
i = 5;                  /* i is now 5  */
i++;                    /* i is now 6  */
++i;                    /* i is now 7 */
i--;                    /* i is now 6 */
--i;                    /* i is now 5 */
```

Placement of Increment/Decrement Operators

SUNG KYUN KWAN UNIVERSITY(SKKU)

- **Different behaviors with different positions**
  - We can use incr/decr operators *before* or *after* a variable, and those two operate in different ways.

Operator	Sample expression	Explanation
++	++a	Increment a by 1, then use the new value of a in the expression in which a resides.
++	a++	Use the current value of a in the expression in which a resides, then increment a by 1.
--	--b	Decrement b by 1, then use the new value of b in the expression in which b resides.
--	b--	Use the current value of b in the expression in which b resides, then decrement b by 1.

Precedence of Arithmetic Operators

SUNG KYUN KWAN UNIVERSITY(SKKU)

- **The rules of operator precedence specify the order C uses to evaluate expressions.**

Operators	Operations	Order of Evaluation (Precedence)
++ --	Increment Decrement	Evaluated first. They cannot be used together at the same time.
( )	Parentheses	Evaluated second. If the parentheses are nested, the expression in the innermost pair is evaluated first. If there are several pairs of parentheses "on the same level" (i.e., not nested), they are evaluated left to right.
* / %	Multiplication Division Remainder	Evaluated third. If there are several, they are evaluated left to right.
+ -	Addition Subtraction	Evaluated last. If there are several, they are evaluated left to right.

C Type Casting

SUNG KYUN KWAN UNIVERSITY(SKKU)

- **Converting between types happens**
- **Two types**
  - Implicit—also called "Automatic". Done FOR you, automatically

```
17 / 5.5
```

» This causes an "implicit type cast" to take place, casting the 17 → 17.0

  - Explicit type conversion

» Programmer specifies conversion with cast operator

```
// same expression as above, using explicit cast
(double) 17 / 5.5
// more typical use; cast operator on variable
(double) myInt / myDouble
```

Increment and decrement operators

DEMO

- Positions of the incr/decr ops affects the results.

```
#include <stdio.h>

int main()
{
        int intNum = 0;

        printf("%d\n", intNum++);
        //printf("%d\n", ++intNum);
        //printf("%d\n", --intNum);
        //printf("%d\n", intNum--);

        return 0;
}
```

Type issues with arithmetic operators

DEMO

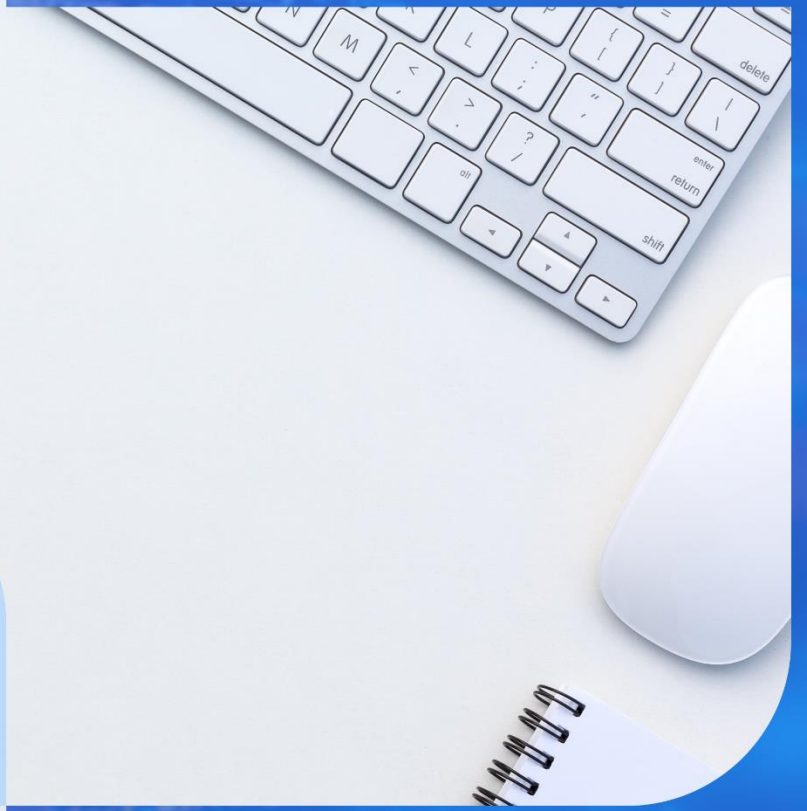
- **Check the effects of implicit type casting. Variations will be shown.**

```
#include <stdio.h>

int main()
{
        int intNum1 = 10;
        int intNum2 = 3;
        int intNum3 = intNum1 / intNum2;
        int intNum4 = intNum1 % intNum2;

        printf("%d\n", intNum3); // What will be the result?
        printf("%d\n", intNum4);

        return 0;
}
```

Basis and Practice in Programming

SUNG KYUN KWAN UNIVERSITY(SKKU)

Programming Basics 1



Instructor Younghoon Kim

Exercise

DEMO

- **Make a program that receives a five-digit number from a user and prints out the digit in the middle of the number. Assume that there are no wrong inputs from a user.**
  - Input: 12345, output: 3
  - Input: 23456, output: 4

```
(intentionally blank box)
```

Exercise

DEMO

- **Make a program that receives a five-digit number from a user and prints out the digit in the middle of the number. Assume that there are no wrong inputs from a user.**
  - Input: 12345, output: 3
  - Input: 23456, output: 4

```
#include <stdio.h>

int main() {
  int num;
  scanf("%d", &num);
  printf("%d\n", num/100%10);

  return 0;
}
```

Placement of ++ and --

- If increment or decrement operators are placed before a variable (i.e., prefixed), they’re referred to as the preincrement or predecrement operators, respectively. Preincrementing (predecrementing) a variable causes the variable to be incremented (decremented) by 1, then the new value of the variable is used in the expression in which it appears.

- If increment or decrement operators are placed after a variable (i.e., postfixed), they’re referred to as the postincrement or postdecrement operators, respectively. Postincrementing (postdecrementing) the variable causes the current value of the variable to be used in the expression in which it appears, then the variable value is incremented (decremented) by 1.

Converting Between Types Explicitly and Implicitly

```
average = ( float ) total / counter;
```

- Often, an average is a value such as 7.2 or –93.5 that contains a fractional part. These values are referred to as floating-point numbers and can be represented by the data type float. The variable average is defined to be of type float to capture the fractional result of our calculation.
- However, the result of the calculation total / counter is an integer because total and counter are both integer variables. Dividing two integers results in integer division in which any fractional part of the calculation is truncated (i.e., lost). Because the calculation is performed first, the fractional part is lost before the result is assigned to average.
- To produce a floating-point calculation with integer values, we must create temporary values that are floating-point numbers. C provides the unary cast operator to accomplish this task.

Converting Between Types Explicitly and Implicitly

```
average = ( float ) total / counter;
```

- includes the cast operator (float), which creates a temporary floating-point copy of its operand, total. The value stored in total is still an integer. Using a cast operator in this manner is called explicit conversion. The calculation now consists of a floating-point value (the temporary float version of total) divided by the unsigned int value stored in counter.

- C provides a set of rules for conversion of operands of different types. Cast operators are available for most data types—they're formed by placing parentheses around a type name. Each cast operator is a unary operator, i.e., an operator that takes only one operand. C also supports unary versions of the plus (+) and minus (-) operators, so you can write expressions such as -7 or +5.

- Cast operators associate from right to left and have the same precedence as other unary operators such as unary + and unary -. This precedence is one level higher than that of the multiplicative operators *, / and %.