


Electrical and Electronic Programming

Professor Mannan Saeed Muhammad
College of Information and Communication Engineering

Class Policy

- **THIS CLASS IS A "NO MOBILE ZONE"**
- Turn OFF your mobile phones …
- If you want to use your mobiles …
  - You can leave the class and return after using it

- If you will use your mobile for chatting or so …
  - You will be ask to leave the class **IMMEDIATELY** …

- **Please don't talk during the lecture to avoid disturbance** …

Course Outline

Topics include (but not limited to) the following:

March

1. Getting Started
2. C Preprocessors
3. Constants, Variables and Data Types
4. Operators and Expressions
5. Decision Making, Branching and Looping
6. User Defined Functions

April

7. Arrays (1D, 2D, 3D /Multi-D)
8. Strings (Character Arrays)

May

9. Structures
10. Unions and Bit Operations

June

11. Pointers
12. File Management and Processing in C

Class Calendar EEE2017-41

- No makeup for Quizzes/Midterm
- Midterm/Quiz date can change according to class progress

- Assignments will be given in class and have to be finished during class time
  - Or if specific time is given

- Late Assignments are not considered

Week	Schedule
1	Intro
2	--
3	--
4	--
5	Quiz 1
6	--
7	--
8	**Midterm / Quiz**
9	--
10	--
11	Quiz 2
12	--
13	--
14	--
15	**Final**

Course Evaluation and Grading Policy

Attendance*	Midterm	Final	Quiz/ CA / Others
25	30	30	15

* 13/15 minimum attendance required : - 8 score for each absent

- < 13/15 → you will be in trouble
- Above distribution can be changed to benefit the class as much as possible

- No take home Assignments
- Only Class Assignments
  - You have to submit the class activities at the end of the class
  - Class activates will be on Computers

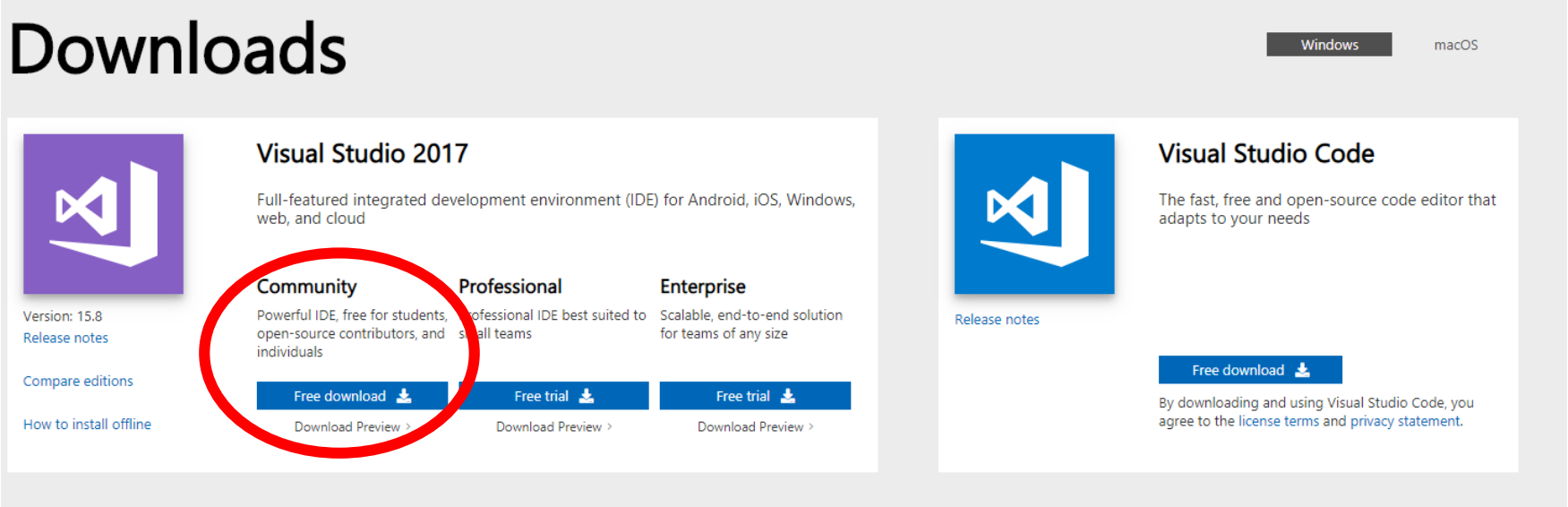
- All lecture files will be uploaded to iCampus/Canvas …

Books

C for Engineers and Scientists An Introduction to Programming with ANSI C,	Gary Bronson
C How to Program,	Paul Deitel, Harvey Deitel
Let us C,	Yashavant P. Kanetkar
Programming in ANSI C,	E. Balagurusamy
Problem Solving and Program Design In C,	Hanly and Koffman

Software In Use

- Microsoft Visual Studio 2019 / 2017 / 2015
  - https://visualstudio.microsoft.com/downloads/



What is Programming?

- ***Programming*** is instructing a computer to do something for you with the help of a language
- The two roles of a programming:
  - **Technical**: It instructs the computer to perform tasks.
  - **Conceptual**: It is a framework within which we organize our ideas about things and processes.
- In programming, we deal with two kind of things:
  - **Data** - representing '*objects*' we want to *manipulate*
  - **Procedures** -'*descriptions*' or '*rules*' that define *how* to manipulate data.

Programming Language

- **Formal Language** used to communicate to a computer.

- A programming language contains ***instructions*** for the computer to perform a specific action or a specific task:
  - 'Calculate the sum of the numbers from 1 to 10‘
  - 'Print “I like programming”
  - ‘Output the current time’

Programming Language

- Can be classified into as a special-purpose and general-purpose programming languages.
- Special-purpose : is design for a particular type of application
  - Structured Query Language (SQL)
- General-purpose : can be used to obtain solutions for many types of problems
  - Machine Languages
  - Assembly Languages
  - High-Level Languages

Machine Language

- The only language that the processor actually 'understands‘
- Consists of binary codes: 0 and 1
  - Example: 00010101
    11010001
    01001100
- Each of the lines above corresponds to a specific task to be done by the ***processor***.
- Programming in machine code is difficult and slow since it is difficult to memorize all the instructions.
- Mistakes can happen very easily.
- Processor and Architecture dependent

Assembly Language

- Enables machine code to be ***represented*** in words and numbers.
- Example of a program in ***assembler language***:

```
LOAD A, 9999
LOAD B, 8282
SUB B
MOV C, A
LOAD C, #0002
DIV A, C
STORE A, 7002
```

- Easier to understand and memorize (called ***Mnemonics***), compared to **machine code** but still quite difficult to use.
- Processor and Architecture dependent

High-Level Language

- Use more English words. They try to resemble English sentences. Therefore, it is easier to program in these languages.
- The programming structure is **problem oriented** - does not need to know how the computer actually ***executes*** the instructions.
- Processor **independent** - the same code can be run on different processors.
- Examples: ***Basic, Fortran, Pascal, Cobol, C, C++, Java***
- A high level language needs to be analyzed by the **compiler** and then compiled into **machine code** so that it can be **executed** by the processor.

C Programming Language

- Why C Language?
  - Most important programming languages
  - Used on every major OS
    - Windows
    - MAC
    - Linux
  - Basis of other languages
    - Objective-C/C++/C#
  - Syntax adapted by other languages
    - C# → Java
  - Used for programming
    - Program applications
    - Compilers
    - OS
    - Even hardware
      - microprocessor/microcontroller based systems
      - Smartphones etc.

C Programming Language

- Based on: ‘A’ (ALGOL), ‘BCPL’ and ‘B’
- Developed by **Dennis Ritchie** at Bell Laboratories in the 1960s
- In cooperation with **Ken Thomson**
  - it was used for Unix systems
- The C Language was only vaguely defined, not standardized, so that almost everyone had his own perception of it, to such an extend that an urgent need for a **standard code** was creeping up

C Programming Language

- In 1983, the American National Standards Institute (ANSI) set up X3J11, a Technical Committee to draft a proposal for the ANSI standard, which was approved in 1989 and referred to as the ANSI/ISO 9899 : 1990 or simply the **ANSI C**, which is now the global standard for **C**.
- This standard was updated in 1999
- In 2007, C11 was introduced

C – An Imperative Language

- C is a highly imperative language
  - We must tell it **exactly how and what** to do;
  - the means and functions to use;
  - which ***libraries*** to use;
  - when to add a new line;
  - when an instruction is finished;

… in short: everything and anything …

General Form of a C Program

```
preprocessor directives
main function heading
{
        declarations
        executable statements
}
```

C Program Structure

- An example of simple program in C

```c
#include <stdio.h>

void main(void)
{
    printf("I love programming\n");
    printf("You will love it too once ");
    printf("you know the trick\n");
    getchar();
}
```

The output

- The previous program will produce the following output on your screen

```
I love programming
You will love it too once you know the trick
```

Preprocessor directives

- a C program line begins with # provides an instruction to the C preprocessor
- It is executed **before** the actual compilation is done.
- Two most common directives :
  - #include
  - #define
- In our example (#include<stdio.h>) identifies the ***header*** file for standard input and output needed by the printf().

☞ The **C Preprocessor** is not part of the compiler, but is a separate step in the compilation process. In simplistic terms, a C Preprocessor is just a text substitution tool and they instruct compiler to do required pre-processing before actual compilation.

Preprocessor directives

Directive	Description
#define	Substitutes a preprocessor macro
#include	Inserts a particular header from another file
#undef	Undefines a preprocessor macro
#ifdef	Returns true if this macro is defined
#ifndef	Returns true if this macro is not defined
#if	Tests if a compile time condition is true
#else	The alternative for #if
#elif	#else an #if in one statement
#endif	Ends preprocessor conditional
#error	Prints error message on stderr
#pragma	Issues special commands to the compiler, using a standardized method

Preprocessor directives

```
#define MAX_ARRAY_LENGTH 20
```

```
#include <stdio.h>
#include "myheader.h"
```

```
#undef  FILE_SIZE
#define FILE_SIZE 42
```

```
#ifndef MESSAGE
   #define MESSAGE "You wish!"
#endif
```

```
#ifdef DEBUG
   /* Your debugging statements here */
#endif
```

```
     #define FIRST

     main()
     {
        int a, b, c;

#ifdef FIRST
       a=2; b=6; c=4;
#else
       printf("Enter a:");
       scanf("%d", &a);

       printf("Enter a:");
       scanf("%d", &a);

       printf("Enter a:");
       scanf("%d", &a);
#endif
       additonal code
```

Standard Predefined Macros

Macro	Description
__DATE__	The current date as a character literal in "MMM DD YYYY" format
__TIME__	The current time as a character literal in "HH:MM:SS" format
__FILE__	This contains the current filename as a string literal.
__LINE__	This contains the current line number as a decimal constant.
__STDC__	Defined as 1 when the compiler complies with the ANSI standard.

```c
#include <stdio.h>

void main(void)
{
    printf("Line : %d\n", __LINE__);
    printf("%s\n",__FILE__);
    printf("%s\n",__DATE__);
    printf("%s\n",__TIME__);
    printf("Line : %d\n\n",__LINE__);
    printf("ANSI : %d\n",__STDC_SECURE_LIB__);
    printf("ANSI : %d\n",__STDC_HOSTED__);
    getchar();
}
```

#define

- You may also associate constant using #define preprocessor directive

```
#include <stdio.h>
#define pi 3.412

void main(void)
{
    double height, radius, base, volume;

    printf("Enter the height and radius of the cone : ");
    scanf_s("%lf %lf", &height, &radius);

    base = pi * radius * radius;
    volume = (1.0 / 3.0) * base * height;

    printf("\nThe volume of a cone is %f\n", volume);
    system("pause");
}
```

Function main

- Identify the start of the program
- Every C program has a main ( )
- 'main' is a C ***keyword***
  - We **must not** use it for any other variable/function
- 4 common ways of main declaration

```
int main(void)
{

    return 0;
}
```

```
void main(void)
{

}
```

```
main(void)
{

}
```

```
main( )
{

}
```

The curly braces { }

- Identify a ***segment / body*** of a program
  - The start and end of a function
  - The start and end of the selection or repetition block.

- Since the opening brace indicates the **start** of a segment with the closing brace indicating the **end** of a segment
  there must be just as many opening braces as closing braces
  (this is a common mistake of beginners)

Statement

- A specification of an action to be taken by the computer as the program executes.
- Each statement in C needs to be terminated with semicolon (;)
- Example:

```
#include <stdio.h>

void main(void)
{
    printf("I love programming\n");
    printf("You will love it too once ");
    printf("you know the trick\n");
    system("pause");
}
```

statement

statement

statement

Statement cont…

- Statement has two types:
  - Declaration
    - The part of the program that tells the compiler the names of memory cells in a program
  - Executable statements
    - Program lines that are converted to machine language instructions and executed by the computer

C program skeleton

- In short, the basic skeleton of a C program looks like this:

```
#include <stdio.h>
void main(void)
{
    statement(s);
}
```

Preprocessor directives

Function main

Start of segment

End of segment

Identifiers

- Words used to represent certain program entities (variables, function names, etc).
- Example:
  - int my_name;
    - my_name is an identifier used as a program variable
  - void CalculateTotal(int value)
    - CalculateTotal is an identifier used as a function name

Rules for Identifiers

- First Character must be an alphabet
- Must consist of only
  letters, digits or underscore(_)
- Only first 31 characters are significant
- Cannot use a keywords
- Must not contain white space

Rules for naming identifiers

Rules	Example
Can contain a mix of characters and numbers. However it **cannot** start with a number	H2o
First character must be a letter or underscore	Number1; _area
Can be of mixed cases including underscore character	XsquAre my_num
Cannot contain any arithmetic operators	R*S+T
… or any other punctuation marks…	#@x%!!
Cannot be a C keyword/reserved word	struct; printf;
Cannot contain a space	My height
… identifiers are **case sensitive**	Tax ≠ tax

Keywords

- Keywords 는 하나의 token으로 간주되며 program에서 특별한 의미를 갖는 예약 어.

ANSI C and ISO C
1970-80

C89, C90 : C++, C95
1990+

C99
Late 1990s

C11 or C1X: C standard revision)
2007 – 2011(Dec-08)

Embedded C
2008

C89 has 32 keywords (reserved words with special meaning):

auto	double	int	struct
break	else	long	switch
case	enum	register	typedef
char	extern	return	union
const	float	short	unsigned
continue	for	signed	void
default	goto	sizeof	volatile
do	if	static	while

C99 adds five more keywords:

_Bool	_Imaginary	restrict
_Complex	inline	

C11 adds seven more keywords:[23]

_Alignas	_Atomic	_Noreturn	_Thread_local
_Alignof	_Generic	_Static_assert	

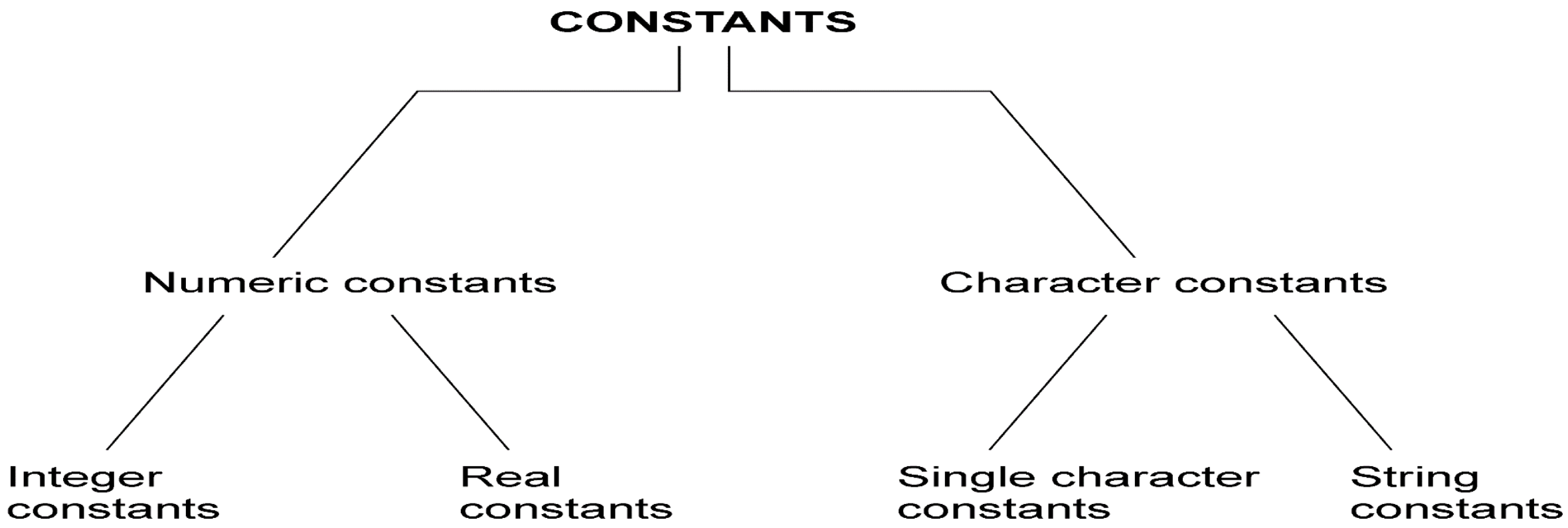
Constants

- Entities that appear in the program code as fixed values
- Any attempt to modify a CONSTANT will result in error
- 4 types of constants:
  - Integer constants
    - Positive or negative whole numbers with no fractional part
    - Example:
      - const int MAX_NUM = 10;
      - const int MIN_NUM = -90;
  - Floating-point constants (float or double)
    - Positive or negative decimal numbers with an integer part, a decimal point and a fractional part
    - Example:
      - const double VAL = 0.5877e2;
        - (stands for 0.5877×10^2)

Constants cont…

- Character constants
  - A character enclosed in a single quotation mark
  - Example:
    - const char letter = ‘n’;
    - const char number = ‘1’;
    - printf(“%c”, ‘S’);
      - Output would be: S
- Enumeration
  - Values are given as a list
  - Example:

```
enum Language {
    Korean,
    English,
    Arabic
};
```



Basic types of C constants

Variables

- **Variable** → a name associated with a memory cell whose <u>value can change</u>

- **Variable Declaration**: specifies the type of a variable
  - Example: `int num;`

- **Variable Definition**: ***assigning*** a value to the declared ***variable***
  - Example: `num = 5`

- Variables (identifier)
  - 변수 – 프로그램 실행 중에 발생할 수 있는 임의의 값(변동되는 값)을 저장하기 위한 기억장소를 말한다.
  - 할당된 memory 주소 대신 변수명에 의해 data저장,참조.
  - 변수의 type에 따라 data가 저장되는 방식과 조작방식이 달라짐으로 type의 설정 중요

C basic data types	
int	Numbers
float	Single precision floating point
double	Double precision floating point
char	Characters

- Variables Declare (변수 선언)
  - 모든 변수는 expression과 statement 내에서 사용되기 전에 반드시 선언하여 compiler에게 필요한 정보 제공

[Ex]

```
int inches, feet, fathoms; /* int type으로 inches, feet,
        fathoms라는 이름의 변수를 선언한다 */

float x, y;        /* float type의 변수 x, y 를 선언한다.  */

char c;            /* character type의 변수 c 를 선언한다. */
```

Variables, Expressions, and Assignments

- Variables Naming Rule
  - 의미 있는 이름을 부여하여 readability를 향상.
  - 영문자 또는 _로 시작. letters, digits, _(underbar) 로 구성.
  - maximum 31자 까지 가능.
  - 예약어는 사용할 수 없음.

[Ex]

사용 가능한 변수명 : times10, get_next_char, _done

사용 불가능한 변수명 : 10times, get-next-char, int

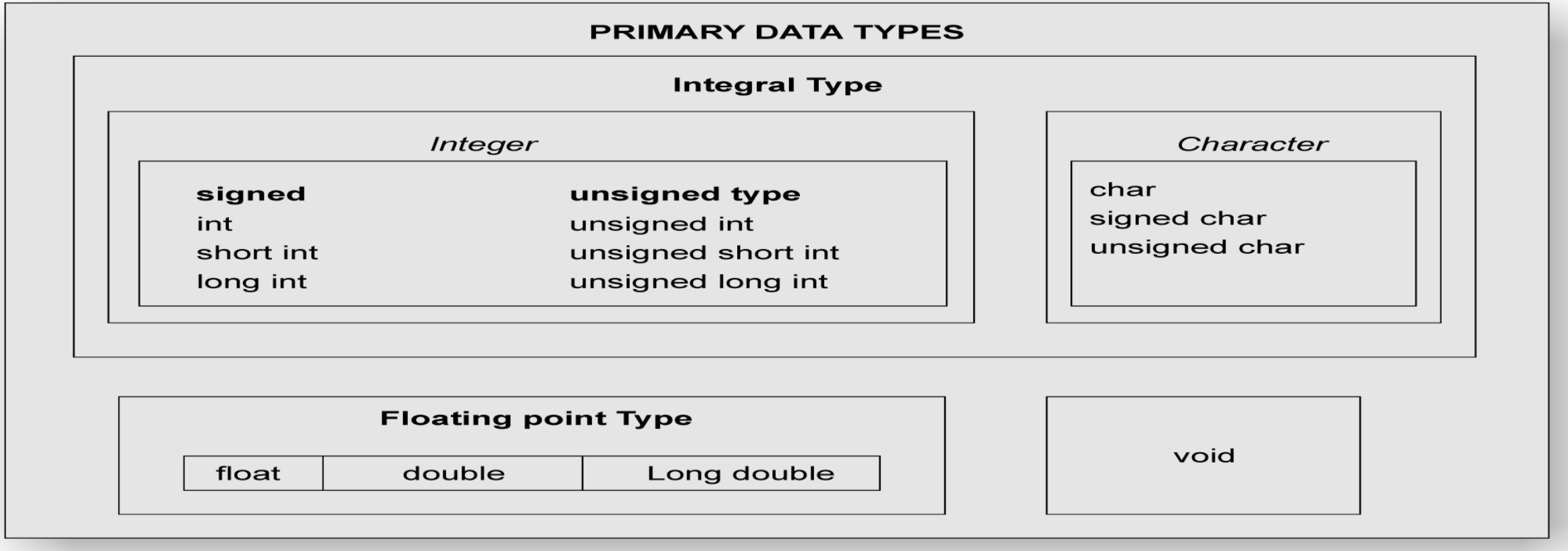
예약어(**Reserved Word**)

auto, break, case, char, const, continue, double

else, enum, float, for, goto, if, int, return, short, signed, sizeof, static, struct, switch, typedef, union, while……

Basic Data Types

Type	Description
char	Typically a single octet(one byte). This is an integer type.
int	The most natural size of integer for the machine.
float	A single-precision floating point value.
double	A double-precision floating point value.
void	Represents the absence of type.



Primary data types in C

Basic Data Types cont…

- There are 4 basic ***data types*** :
  - int
  - float
  - double
  - char
- **int** -32,768 ~ 32,767 (16bit @ 16bit machine)
  - used to declare numeric program variables of integer type
  - whole numbers, positive and negative
  - keyword: **int**

```
int number;
number = 12;
```

Integer type

Type	Storage size	Value range
char	1 byte	-128 to 127 or 0 to 255
unsigned char	1 byte	0 to 255
signed char	1 byte	-128 to 127
int	2 or 4 bytes	-32,768 to 32,767 or -2,147,483,648 to 2,147,483,647
unsigned int	2 or 4 bytes	0 to 65,535 or 0 to 4,294,967,295
short	2 bytes	-32,768 to 32,767
unsigned short	2 bytes	0 to 65,535
long	4 bytes	-2,147,483,648 to 2,147,483,647
unsigned long	4 bytes	0 to 4,294,967,295

Basic Data Types cont…

3.4e-38 ~ 3.4e38 (32bit @ 16bit machine)

- **float**
  - fractional parts, positive and negative
  - keyword: **float**

    float height;

    height = 1.72;

1.7e-308 ~ 1.7e308 (64bit @ 16bit machine)

- **double**
  - used to declare floating point variable of higher precision or higher range of numbers
  - exponential numbers, positive and negative
  - keyword: **double**

    double valuebig;

    valuebig = 12E-3;

Float type

Type	Storage size	Value range	Precision
float	4 byte	1.2E-38 to 3.4E+38	6 decimal places
double	8 byte	2.3E-308 to 1.7E+308	15 decimal places
long double	10 byte	3.4E-4932 to 1.1E+4932	19 decimal places

Basic Data Types cont…

float

double

Basic Data Types cont…

-128 ~ 127(8bit @ 16bit machine)

- **char**
  - equivalent to ‘letters’ in English language
  - Example of characters:
    - Numeric digits: 0 - 9
    - Lowercase/uppercase letters: a - z and A - Z
    - Space (blank)
    - Special characters: , . ; ? “ / ( ) [ ] { } * & % ^ < > etc
  - single character
  - keyword: **char**

```
char my_letter;
my_letter = 'U';
```

The declared character must be enclosed within a single quote!

- In addition, there are **void, short, long,** etc.

```
main() /*.........Program Name........................ */
{
  /*................Declaration.......................*/
    float       x, y;
    int         code;
    short int   count;
    long int    amount;
    double      deviation;
    unsigned    n;
    char        c;
  /*................Computation.......................  */
    . . . .
    . . . .
    . . . .
} /*.............Program ends........................*/
```

Declaration of variables

Void type

S.N.	Types and Description
1	**Function returns as void** There are various functions in C which do not return value or you can say they return void. A function with no return value has the return type as void. For example **void exit (int status);**
2	**Function arguments as void** There are various functions in C which do not accept any parameter. A function with no parameter can accept as a void. For example, **int rand(void);**
3	**Pointers to void** A pointer of type void * represents the address of an object, but not its type. For example a memory allocation function **void *malloc(size_t size);** returns a pointer to void which can be casted to any data type.

※ Example: variable type

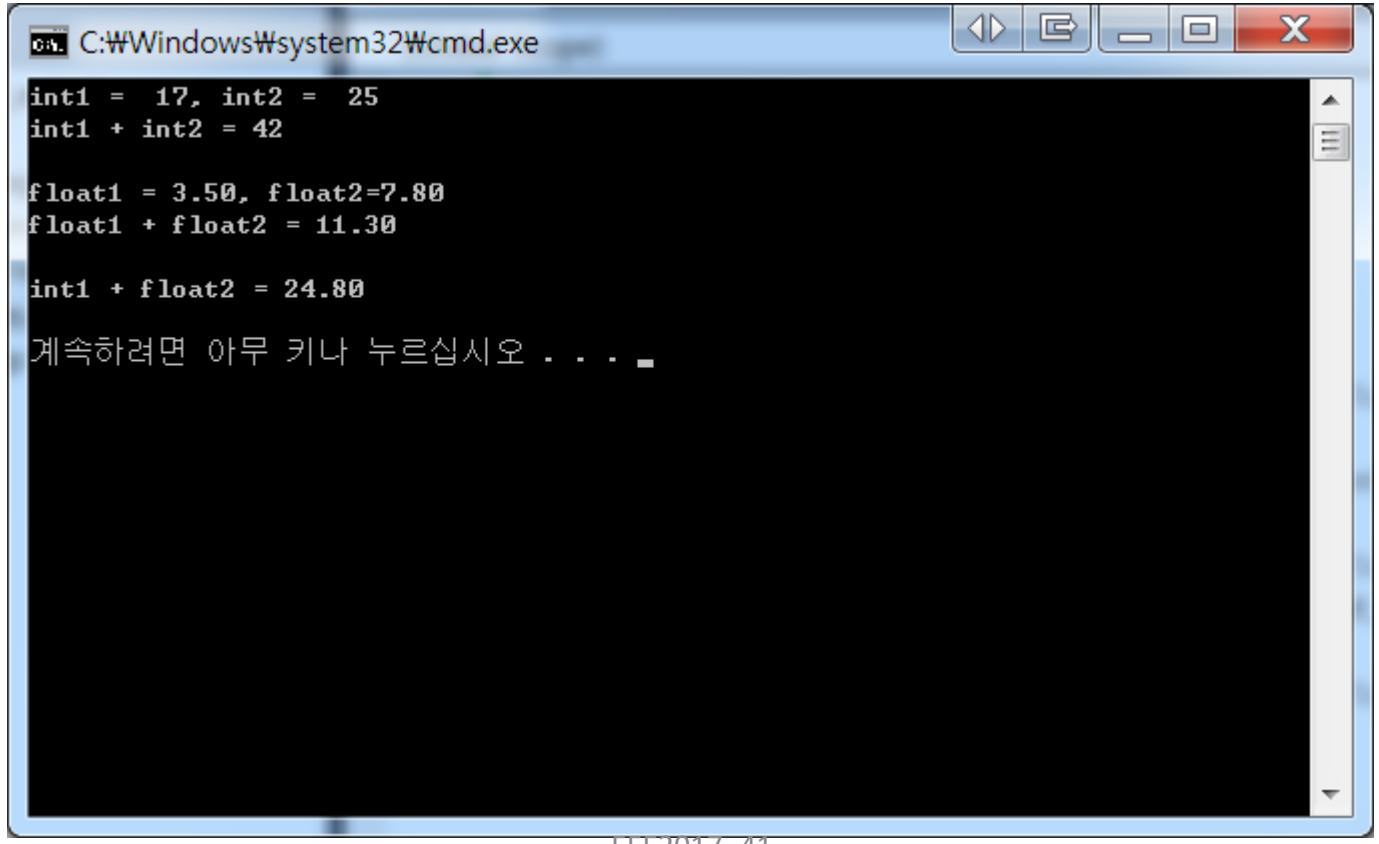
- This example shows the result which sum of two variable on the screen.

```c
#include<stdio.h>

int main(void)
{
    int int1, int2, int_sum;                        //declare int type variable
    float float1, float2, float_sum;                //declare float type variable
    int1=17; int2=25; int_sum=int1+int2;
    float1=3.5; float2=7.8; float_sum=float1+float2;

    printf("int1 = %3d, int2 = %3d₩n", int1, int2);
    printf("int1 + int2 = %d₩n₩n",int_sum);         //int1+int2
    printf("float1 = %3.2f, float2=%3.2f₩n", float1, float2);
    printf("float1 + float2 = %3.2f₩n₩n", float_sum);   //float1+float2

    printf("int1 + float2 = %3.2f₩n₩n", int1+float2);   //int+float2
    return 0;
}
```

```c
#include<stdio.h>

int main(void)
{
    int int1, int2, int_sum;                        //declare int type variable
    float float1, float2, float_sum;                //declare float type variable
    int1=17; int2=25; int_sum=int1+int2;            //define int type variable value
    float1=3.5; float2=7.8; float_sum=float1+float2;   //define float type variable value

    printf("int1 = %3d, int2 = %3d₩n", int1, int2);
    printf("int1 + int2 = %d₩n₩n",int_sum);         //int1+int2
    printf("float1 = %3.2f, float2=%3.2f₩n", float1, float2);
    printf("float1 + float2 = %3.2f₩n₩n", float_sum);  //float1+float2

    printf("int1 + float2 = %3.2f₩n₩n", int1+float2);  //int+float2
    return 0;
}
```

※ Example: variable type

- This example shows the result which sum of two variable on the screen.



Program

```
main()
{
/*..........DECLARATIONS.............................*/
     float     x,p ;
     double    y,q ;
     unsigned  k ;
/*..........DECLARATIONS AND ASSIGNMENTS............*/
     int       m = 54321 ;
     long int  n = 1234567890 ;
/*..........ASSIGNMENTS.............................*/
     x = 1.234567890000 ;

     y = 9.87654321 ;
     k = 54321 ;
     p = q = 1.0 ;
/*..........PRINTING................................*/
     printf("m = %d\n", m) ;
     printf("n = %ld\n", n) ;
     printf("x = %.12lf\n", x) ;
     printf("x = %f\n", x) ;
     printf("y = %.12lf\n",y) ;
     printf("y = %lf\n", y) ;
     printf("k = %u p = %f q = %.12lf\n", k, p, q) ;
}
```

Output

```
m = -11215
n = 1234567890
x = 1.234567880630
x = 1.234568
y = 9.876543210000
y = 9.876543
k = 54321 p = 1.000000 q = 1.000000000000
```

Examples of assignments