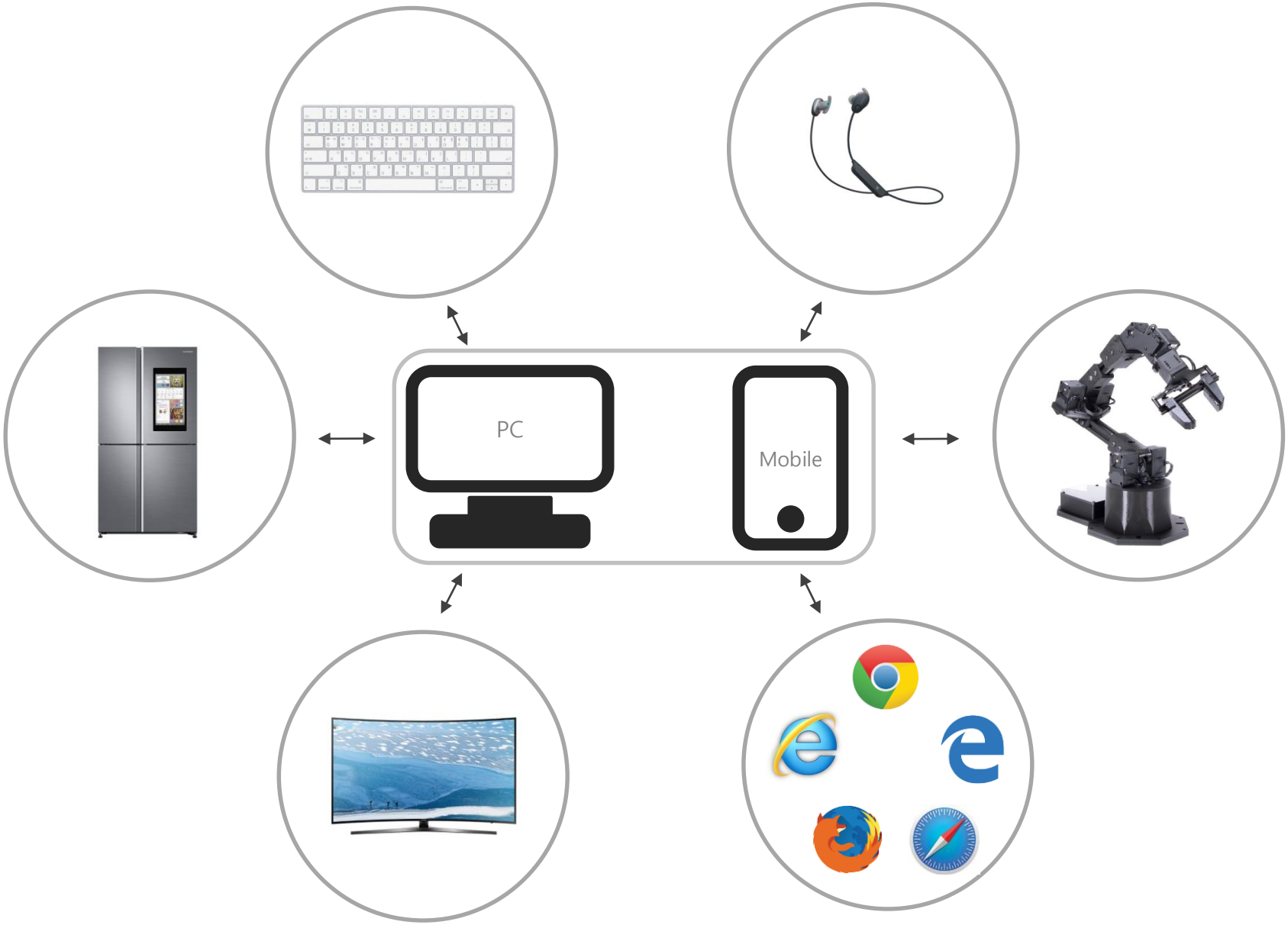
C 언어 교육 자료

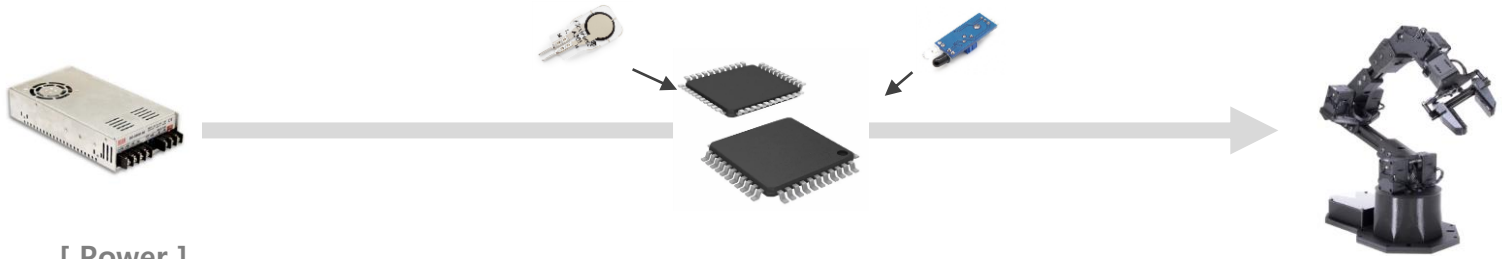
목차

Chapter 1. 프로그램으로 무엇을 하는가?



Chapter 1. 프로그램은 어떻게 사용되는가?

[Power] [Switch] [전구]



[Power] [Machine language] [Program] [로봇]

[Power] [Machine language] [Program] [Data Control]

Chapter 1. 글씨와 그림은 어떻게 화면에 보일까?

[Compile]

[Coding]

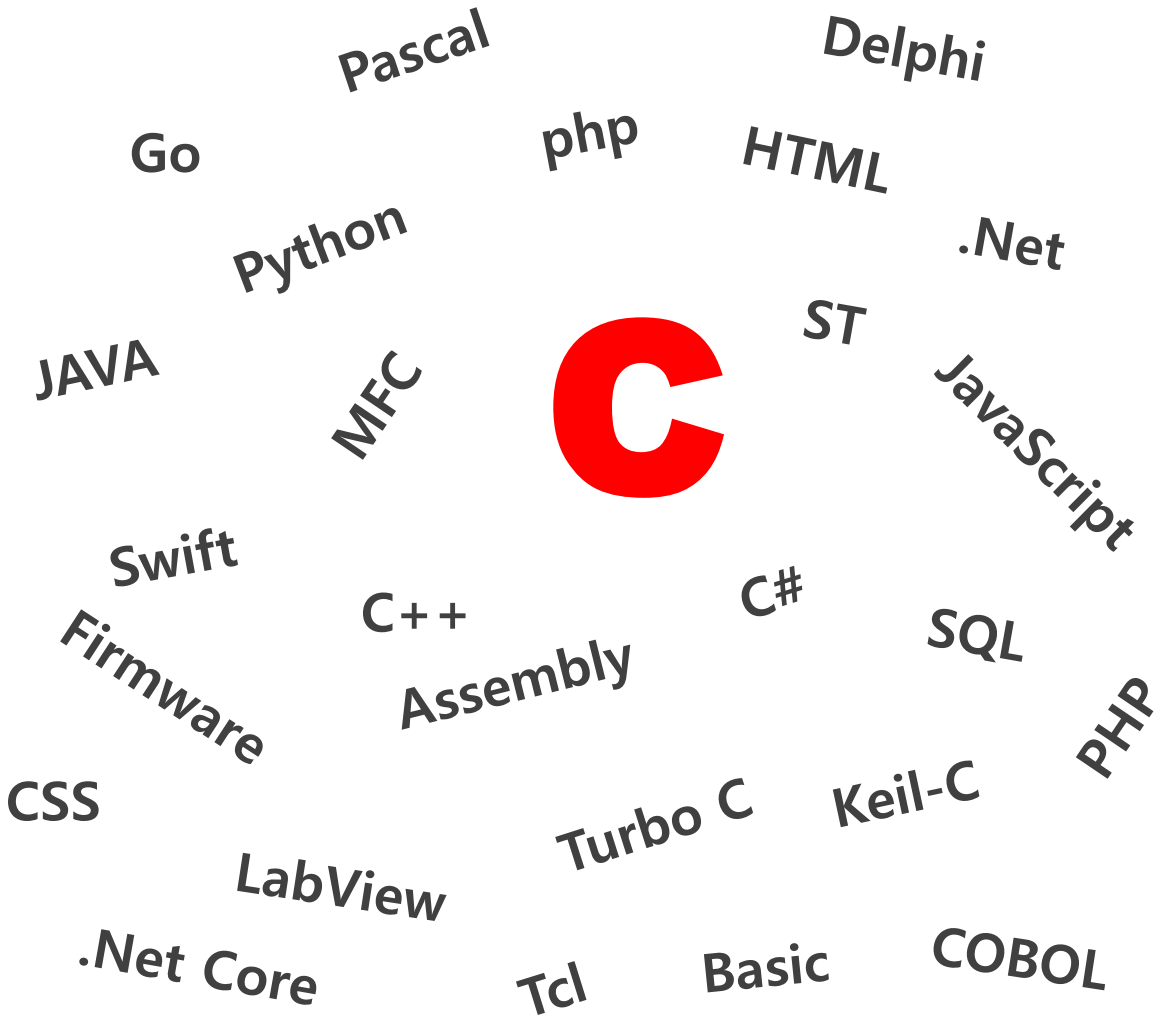
[Programmer]

[Machine Language]

[MCU]

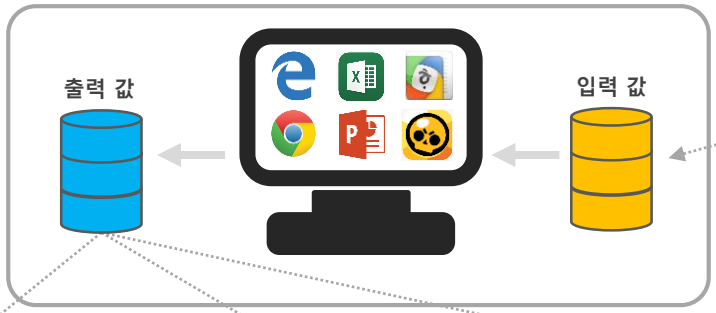
[Output]

Chapter 1. 어떤 언어를 배울 것인가?

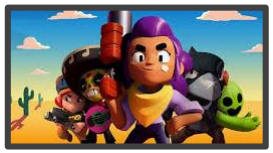


- **Web Browser**
  - Java Script
  - Python
  - HTML
  - PHP
- **Device Control**
  - C, C++, C#, MFC
  - Delphi
- **DB Control**
  - SQL
- **Application**
  - MFC, C#, JAVA, Delphi

Chapter 2. 변수는 왜 필요할까?



[데이터 입력]

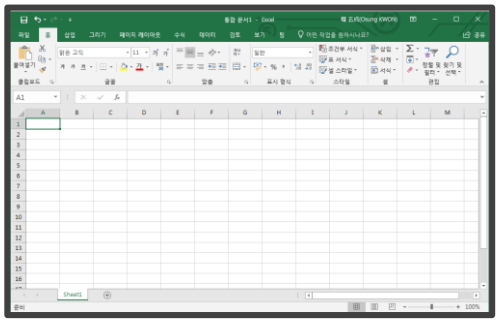


[게임 조작]

Google YouTube

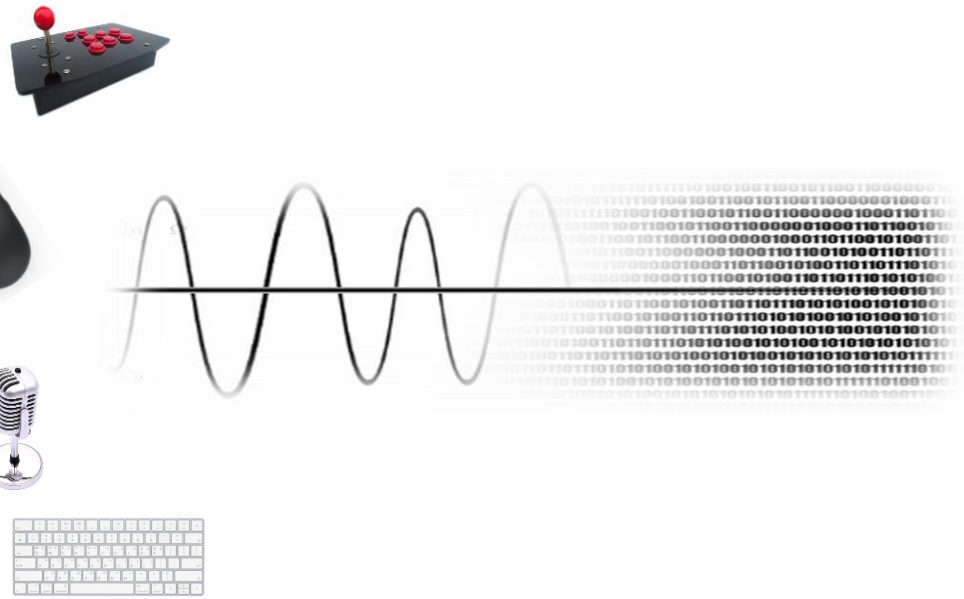
blog

[Web]



[문서 작업]

Chapter 2. 변수에는 어떤 종류가 있을까?



- **문자**
  - 한 글자 (Char : Character)
  - 문장 (String)
    - → Multibyte [1 Byte에 한 글자]
    - → Unicode [다수 Byte에 한 글자]
- **숫자**
  - 정수 (short , long, int : Integer)
  - 자연수 (unsigned int)
  - 실수 (float, double)

[1 MByte]

[1,000,000 Byte]

[1 GByte]

[1,000,000,000 Byte]

[1 TByte]

[1,000,000,000,000 Byte]

- **BIT**
- **BYTE**

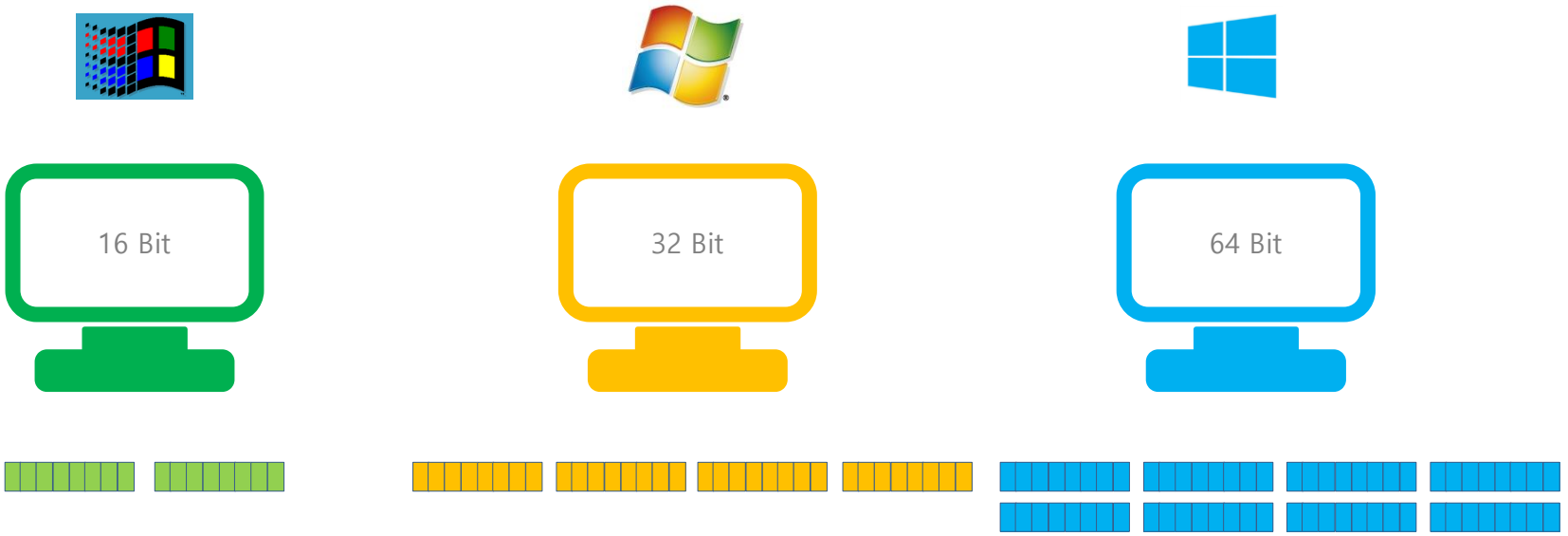
[1 Byte = 8 Bit]

Chapter 2. 변수는 어떤 특성이 있을까?

1) 비교는 같은 모양의 집끼리 해야 한다.
2) 큰집 살던 사람은 작은 집으로 가면 못살 수 있다.
3) 주소를 통해 집을 찾을 수 있다.
4) 집에 들어갈 수 있는 사람의 최대 수는 정해져 있다.
5) 주소는 집의 크기 만큼씩 차이가 발생한다.
6) 집이 변수이다.
7) 땅이 메모리이다.
8) 부동산이 OS이다.

* 주소 – 포인터
* 크기 – 사이즈
* 모양 – Type
* 사람 – Data

Chapter 2. 변수의 크기는 항상 동일한가?



[int]

2 Byte
-32768 ~ +32767

$-2^{15} \sim +(2^{15}-1)$

4 Byte
-2147483648 ~ +2147483647

$-2^{31} \sim +(2^{31}-1)$

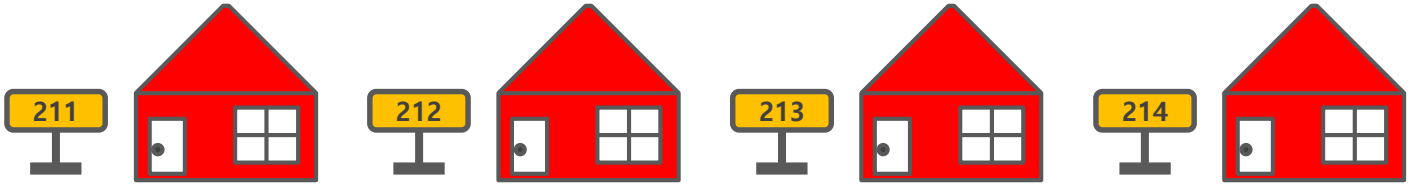
8 Byte
-9223372036854775808 ~ +9223372036854775807

$-2^{63} \sim +(2^{63}-1)$

Chapter 2. 변수 표현 방법 (1)

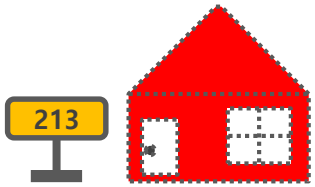
byte Value;

int Value = 5;

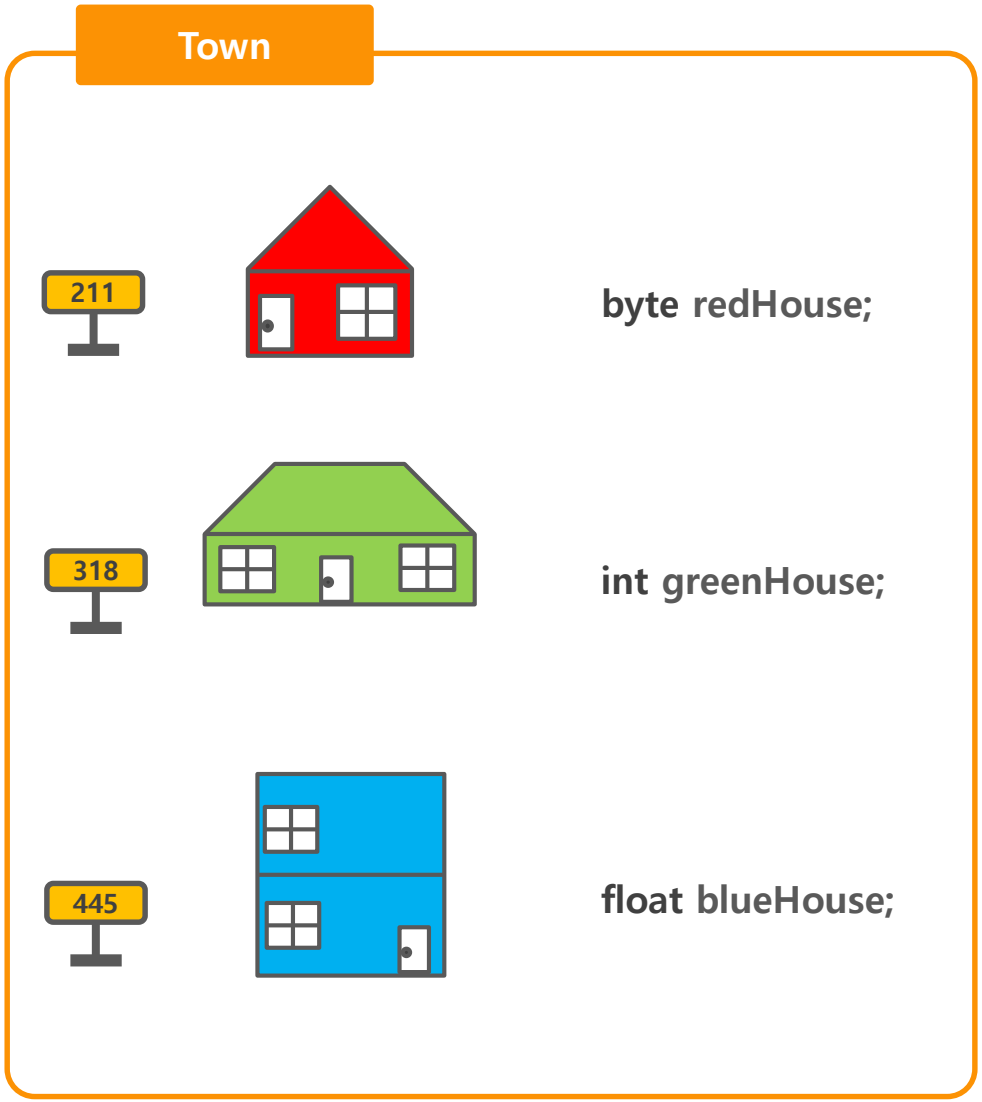


byte Value[4];

byte *Value;



byte *Value = new byte;

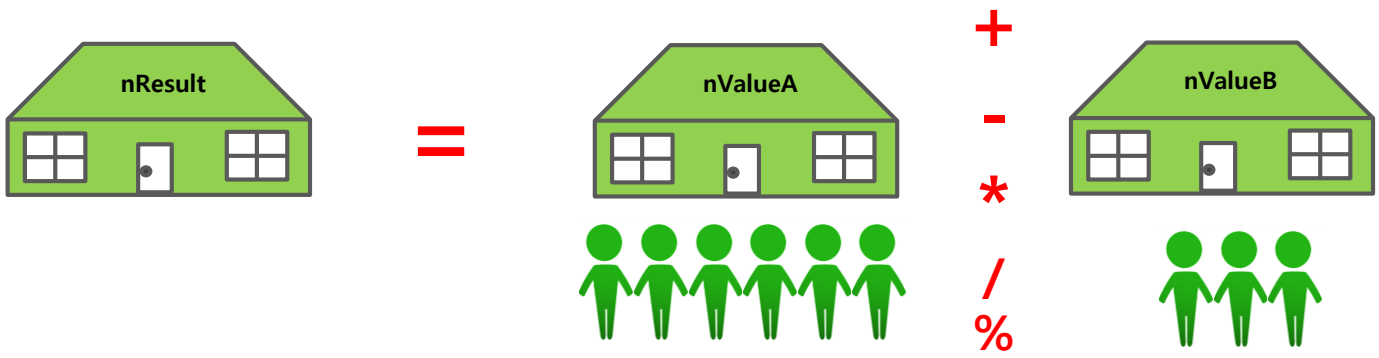



```
struct town
{
    byte redHouse;
    int greenHouse;
    float blueHouse;
};
```

Chapter 3. 연산자의 구분

산술 연산자		비교(관계) 연산자		논리 연산자		비트 연산자	
덧셈	+	같음	==	부정 (NOT)	!	부정 (NOT)	~
뺄셈	-	같지 않음	!=	~이고 (AND)	&&	~이고 (AND)	&
곱셈	*	큼	>	~이거나 (OR)	||	~이거나 (OR)	|
나눗셈	/	작음	<			같지 않으면(XOR)	^
모듈러	%	크거나 같음	>=			왼쪽으로 이동	<<
증가	++	작거나 같음	<=			오른쪽으로 이동	>>
감소	--						

- 치환(직접할당) A = B; B의 값을 A에 넣어 준다.



[숫자 값 16]

```
byte ValueA = 16;
byte ValueA = 0b00010000;
byte ValueA = 020;
byte ValueA = 0x10;
```

0	0	0	1	0	0	0	0
2^7	2^6	2^5	2^4	2^3	2^2	2^1	2^0
128	64	32	16	8	4	2	1

0x10

0	0	0	1
8	4	2	1

0	0	0	0
8	4	2	1

0x1 0

byte ValueA = 1;

byte ValueB = 3;

OR Result;

AND Result;

XOR Result;

ValueA << 2 Result;

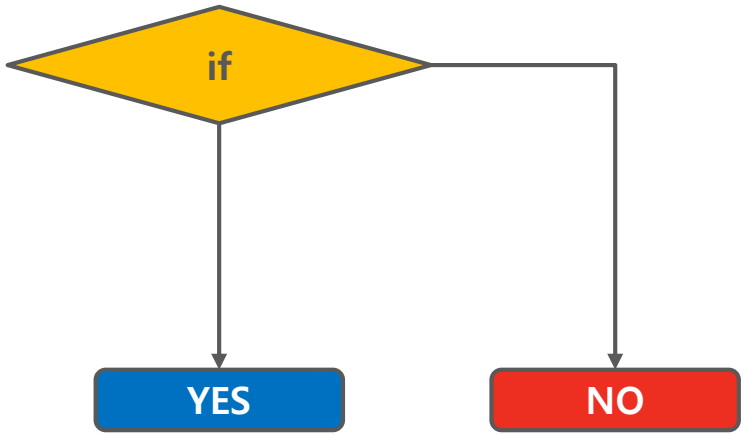
ValueB << 3 Result;

Chapter 4. 조건 문 (if)

if: 만약 ~ 라면

비교(관계) 연산자		논리 연산자	
같음	==	부정 (NOT)	!
같지 않음	!=	~이고 (AND)	&&
큼	>	~이거나 (OR)	||
작음	<		
크거나 같음	>=		
작거나 같음	<=		

[Check]




```
int nValueA, nValueB, nValueC, nValueD;
```

```
if ( nValueA == nValueB )  nValueC = 3;
```

```
if ( nValueA == nValueB )
   nValueC = 3;
   nValueC = 9;
```

```
if ( nValueA == nValueB )  nValueC = 3;
if ( nValueA == nValueB )  nValueD = 9;
```

```
if ( nValueA == nValueB )
{
   nValueC = 3;
   nValueD = 9;
}
```

Chapter 4. 조건 문 (if else)

if: 만약 ~ 라면, **else:** ~다른 것은?

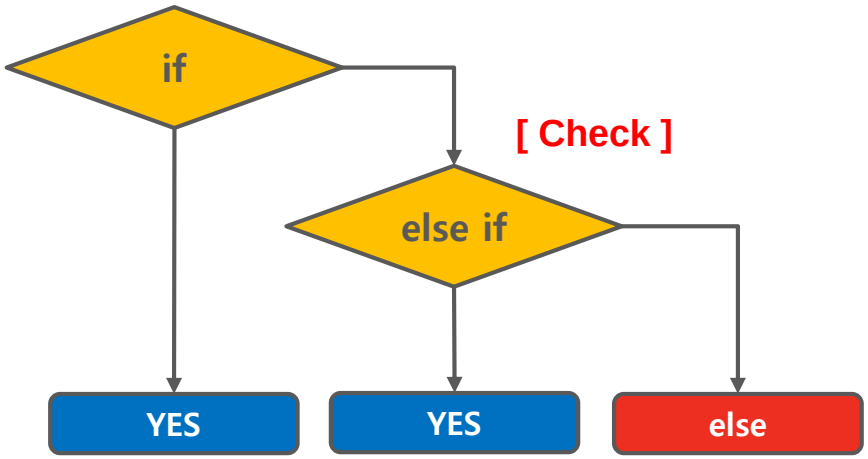
```
int nValueA, nValueB, nValueC, nValueD;
```

```
if ( nValueA == nValueB )  nValueC = 3;
else                       nValueC = 4;
```

```
if ( nValueA == nValueB )        nValueC = 3;
else if ( nValueA > nValueB )    nValueC = 4;
else                             nValueC = 5;
```

```
if ( nValueA == nValueB ) {
   nValueC = 3;
   nValueD = 9;
}
else {
   nValueC = 7;
   nValueD = 9;
}
```

[Check]




```
if ( nValueA == nValueB ) {
   nValueC = 3;
   nValueD = 9;
}
else if ( nValueA > nValueB ) {
   nValueC = 4;
   nValueD = 6;
}
else {
   nValueC = 7;
   nValueD = 9;
}
```

Chapter 4. 조건 문 (switch)

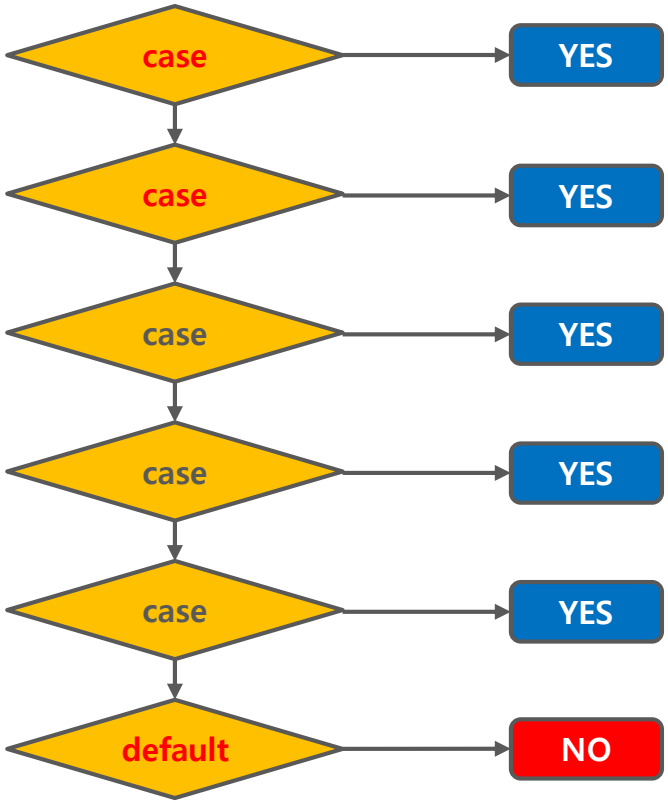
switch: 전환 , **case:** ~경우

```
int nValueA, nValueB, nValueC;
switch( nValue )
{
  case 1:     nValueB = 3;
    break;
  case 2:     nValueB = 9;
    break;
  case 3:     nValueB = 2;
    break;
  case 4:     nValueB = 4;
    break;
  case 5:     nValueB = 5;
    break;
  default:    nValueB = 8;
    break;
}
```

```
switch( nValue )
{
  case 3:  nValueB = 3;
  case 4:  nValueB = 2;
  case 7:  nValueB = 4;
  default: nValueB = 8;
}
```

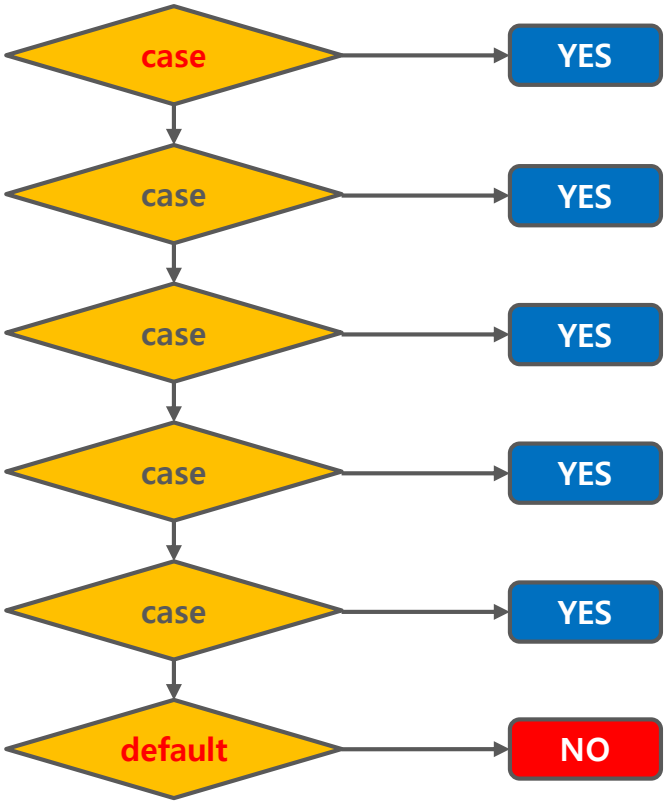
```
switch( nValue )
{
  case 4:
    nValueB = 3;
    nValueC = 1;
   break;
   ..............
}
```

[Select]

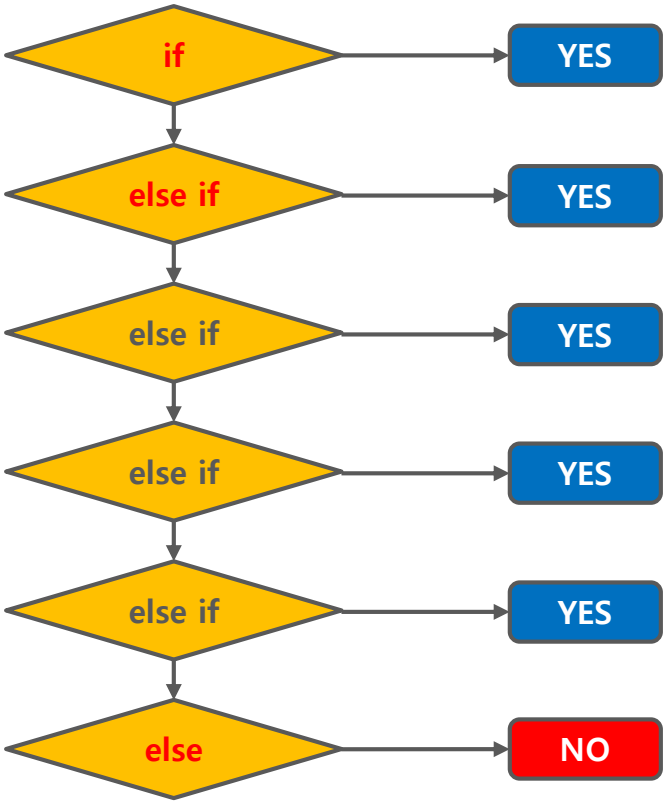


Chapter 4. 조건 문 (if else - switch)

[Select]



[Check]

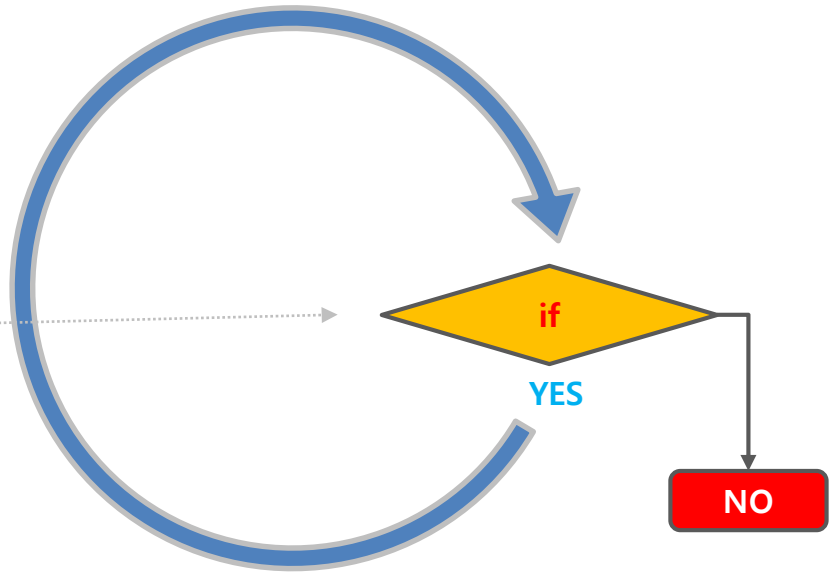


Chapter 5. 반복 문 (While)

while: ~하는 동안

```
int nValueA;
nValueA = 1;

while( nValueA < 10 )
{
   nValueA++;
}
```

[while]

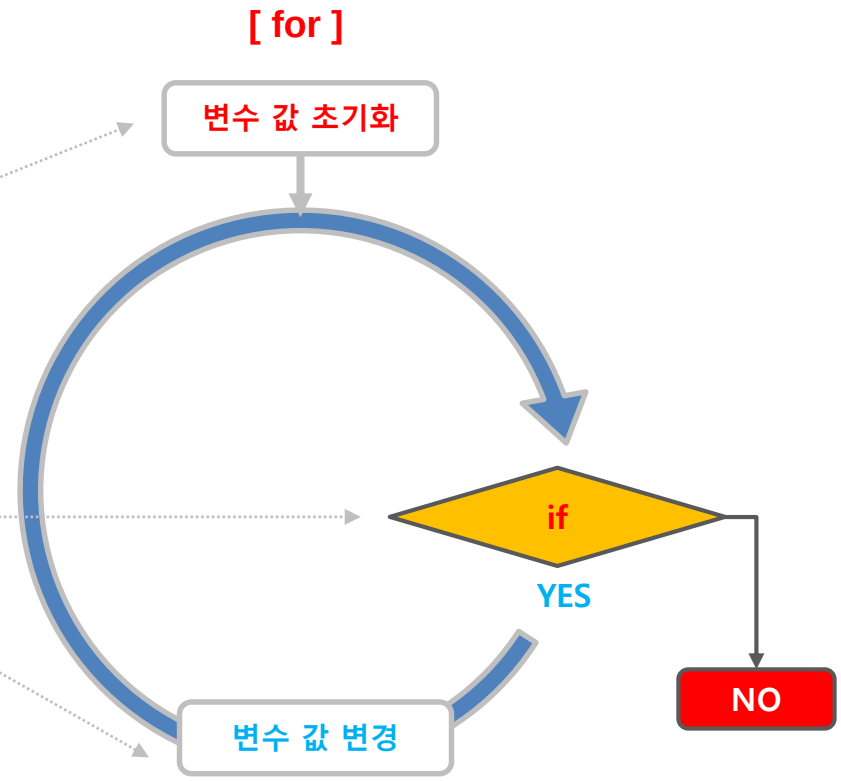


무한 루프 (infinite loop)	while (1)
	{
	.....
	}

Chapter 5. 반복 문 (for)

for: ~하기 위하여

```
int nValueA, nValueB, nValueC;

nValueB = 0;
nValueC = 0;

for( nValueA = 0 ; nValueA <= 9 ; nValueA++ )
{
      nValueB = nValueA + 1;
      nValueC = nValueC + nValueB;
}
```



무한 루프 (infinite loop)	for (; 1 ;)
	{
	.....
	}

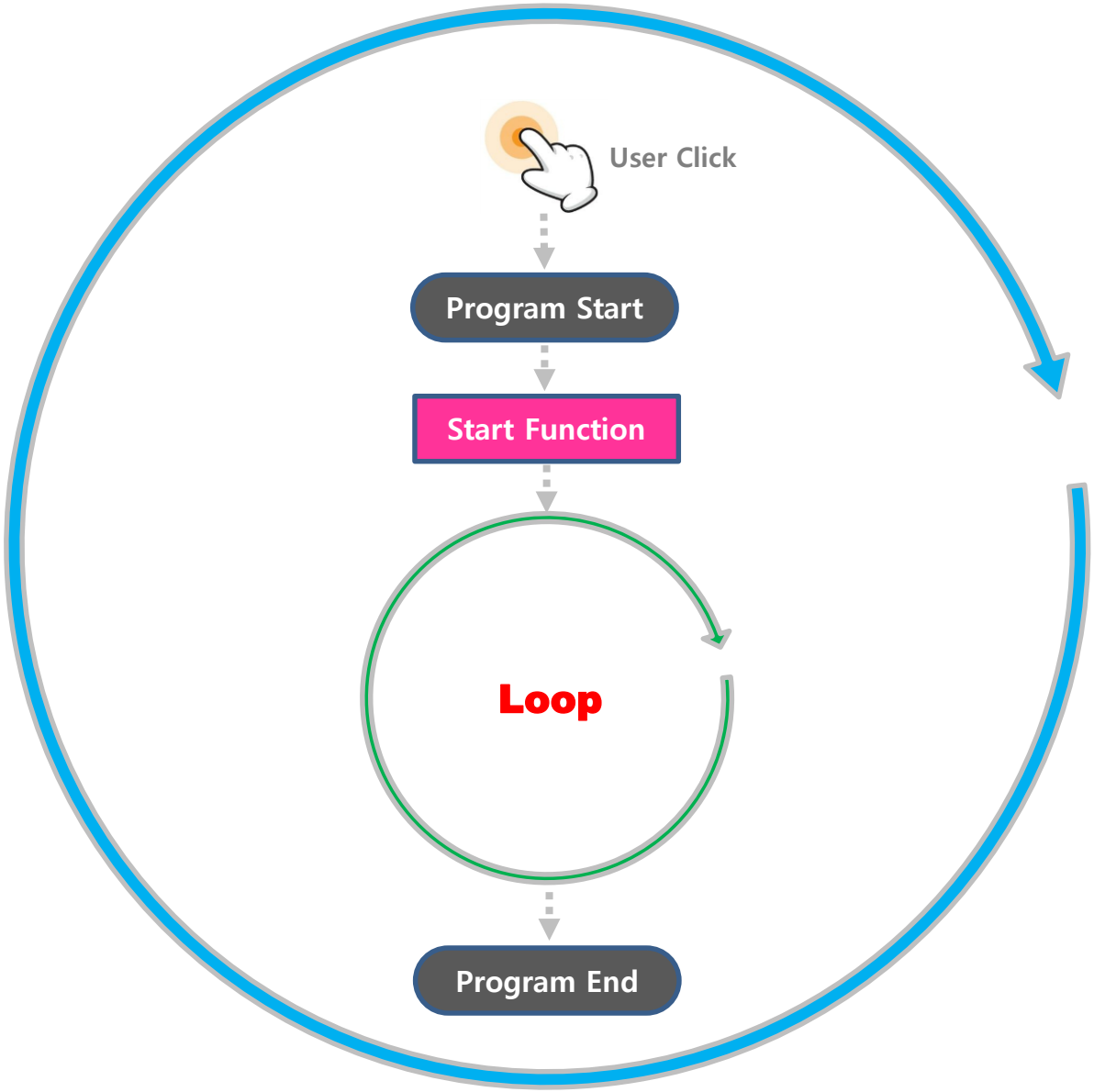
1~ 10 까지의 합을 구하는 코드 비교

```
int nValueA, nValueB, nValueC;

nValueA = 1;
nValueB = 0;
nValueC = 0;

while( nValueA <= 10 )
{
    nValueB = nValueA;
    nValueC = nValueC + nValueB;
    nValueA++;
}
```

```
int nValueA, nValueB, nValueC;

nValueB = 0;
nValueC = 0;

for( nValueA = 1 ; nValueA <= 10 ; nValueA++ )
{
    nValueB = nValueA;
    nValueC = nValueC + nValueB;

}
```

프로그램은 어떻게 시작되고 돌아갈까?

[OS : Window, Linux...]



Chapter 6. 함수란 무엇인가?

함수: 기능

```
int nValueA, nValueB, nValueSum;
```

```
nValueA = 3;
nValueB = 4;
nValueSum = nValueA + nValueB;
```

```
nValueA = 2;
nValueB = 5;
nValueSum = nValueA + nValueB;
```

```
nValueA = 7;
nValueB = 9;
nValueSum = nValueA + nValueB;
```

```
nValueSum = nValueA + nValueB;
```

```
int SumFunction( int _valueA, int _valueB )
{
  int nSum = 0;
  nSum = _valueA + _valueB;

  return nSum;
}
```

함수 (Function, Method)란 반복되는 기능을 모아 놓은 묶음이다.

Chapter 6. 함수의 리턴(return), 인자(argument), 매개변수(parameter)

return: 돌아오다, **argument:** 사물의 원인이나 요소

```
int nValueSum = 0;

nValueSum = SumFunction( 3 , 4 );
```

함수 호출 / 인자

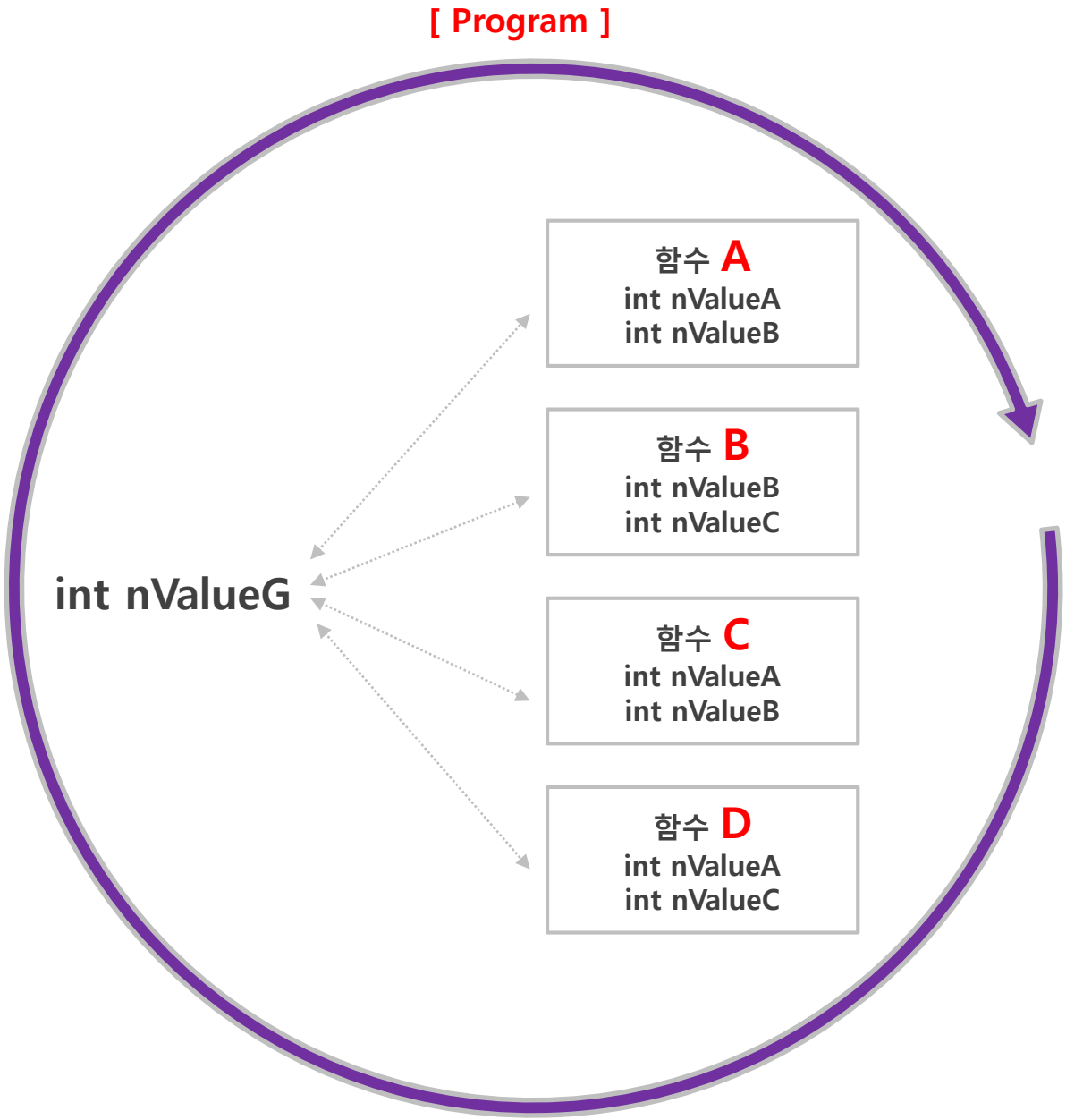
```
int SumFunction( int _valueA, int _valueB )
{
  int nSum = 0;
  nSum = _valueA + _valueB;
  return nSum;
}
```

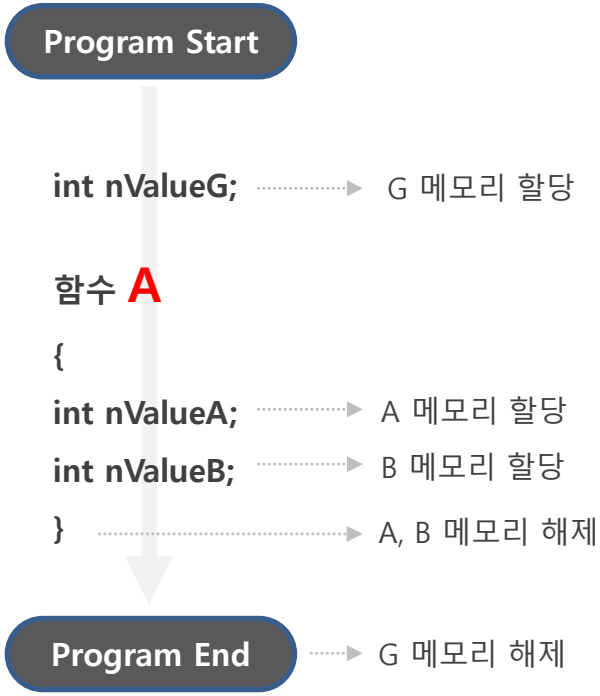
return Type / parameter / 반환 코드

void : 빈 공간, ~이 하나도 없는
반환 값이 없는 경우에 사용한다.
void DisplayResult() → return 코드가 존재 하지 않는다.

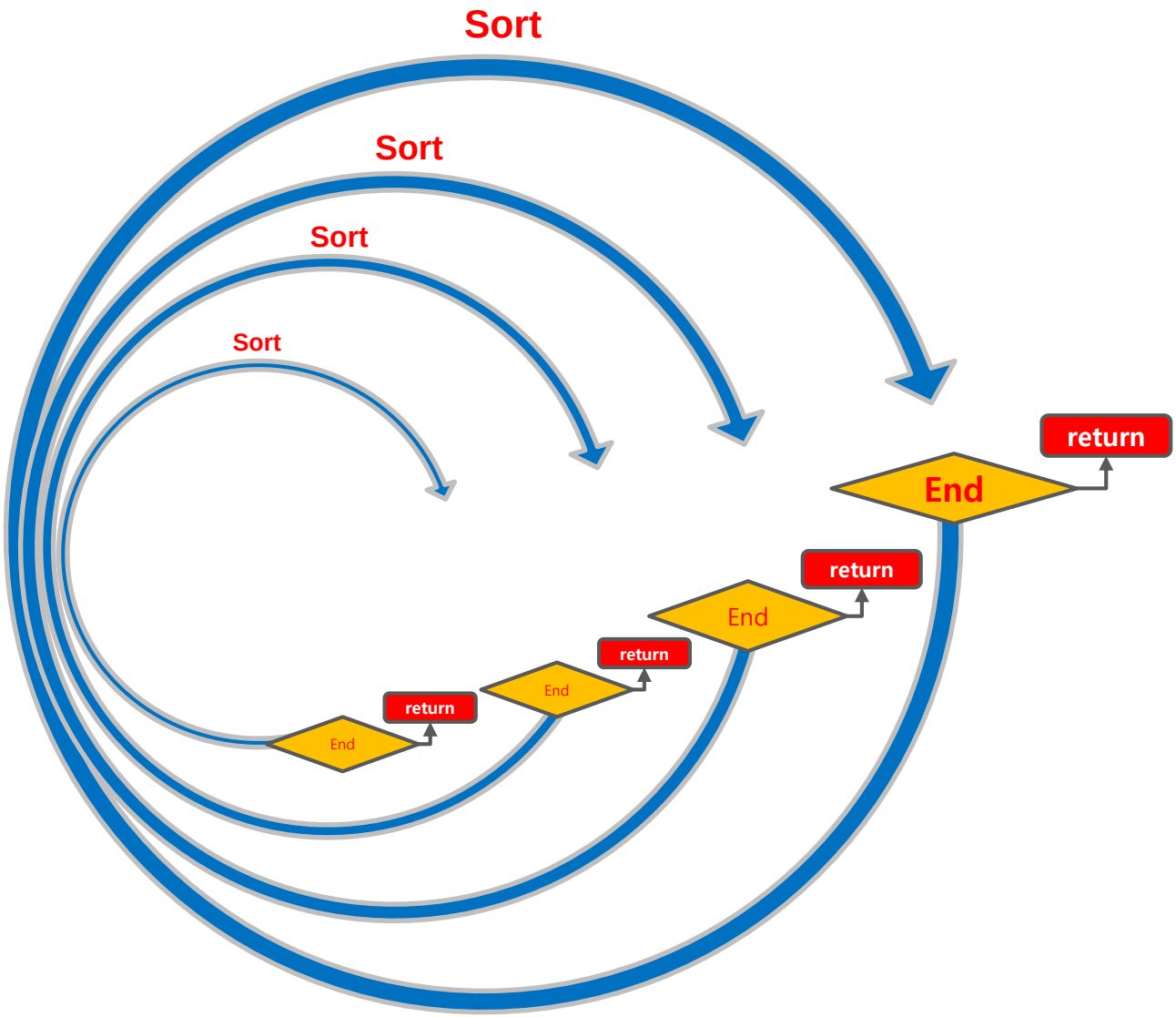
```
int SumFunction( int _valueA, int _valueB ) {
  float result = (float) ( _valueA + _valueB );
}
```

Chapter 7. 전역 (Global) 변수 와 지역(Local) 변수





재귀 (recursion) : 어떠한 것을 정의할 때 자기 자신을 참조하는 것.




```
void Sort()
{
    // Code ......

    if( A > B ) Sort();

    // Code .......

}
```

.....

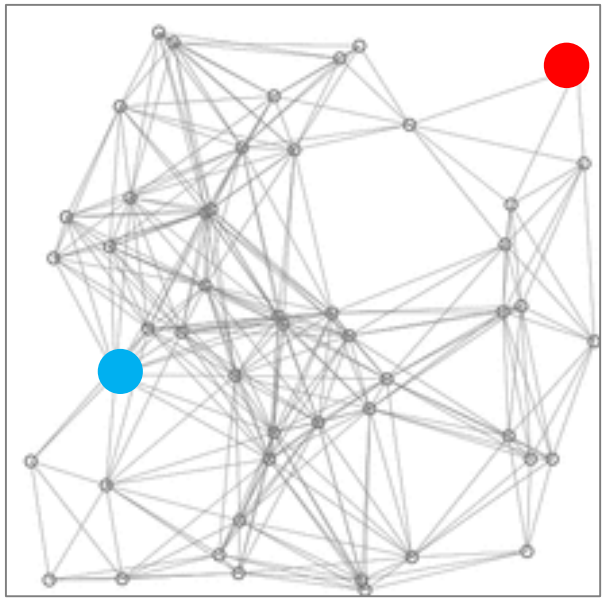
Chapter 8. 재귀 함수는 어떠한 경우에 사용하는가?

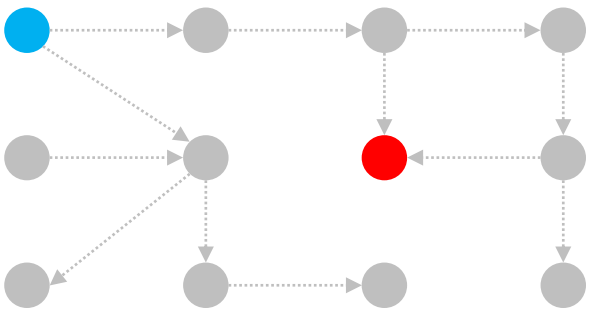
Sort [정렬]

a[i]

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
M	E	R	G	E	S	O	R	T	E	X	A	M	P	L	E
E	M	R	G	E	S	O	R	T	E	X	A	M	P	L	E
E	M	G	R	E	S	O	R	T	E	X	A	M	P	L	E
E	M	G	R	E	S	O	R	T	E	X	A	M	P	L	E
E	M	G	R	E	S	O	R	T	E	X	A	M	P	L	E
E	M	G	R	E	S	O	R	E	T	X	A	M	P	L	E
E	M	G	R	E	S	O	R	E	T	A	X	M	P	L	E
E	M	G	R	E	S	O	R	E	T	A	X	M	P	L	E
E	M	G	R	E	S	O	R	E	T	A	X	M	P	E	L
E	G	M	R	E	S	O	R	E	T	A	X	M	P	E	L
E	G	M	R	E	O	R	S	E	T	A	X	M	P	E	L
E	G	M	R	E	O	R	S	A	E	T	X	M	P	E	L
E	G	M	R	E	O	R	S	A	E	T	X	E	L	M	P
E	E	G	M	O	R	R	S	A	E	T	X	E	L	M	P
E	E	G	M	O	R	R	S	A	E	E	L	M	P	T	X
A	E	E	E	E	G	L	M	M	O	P	R	R	S	T	X

Path Find [길 찾기]

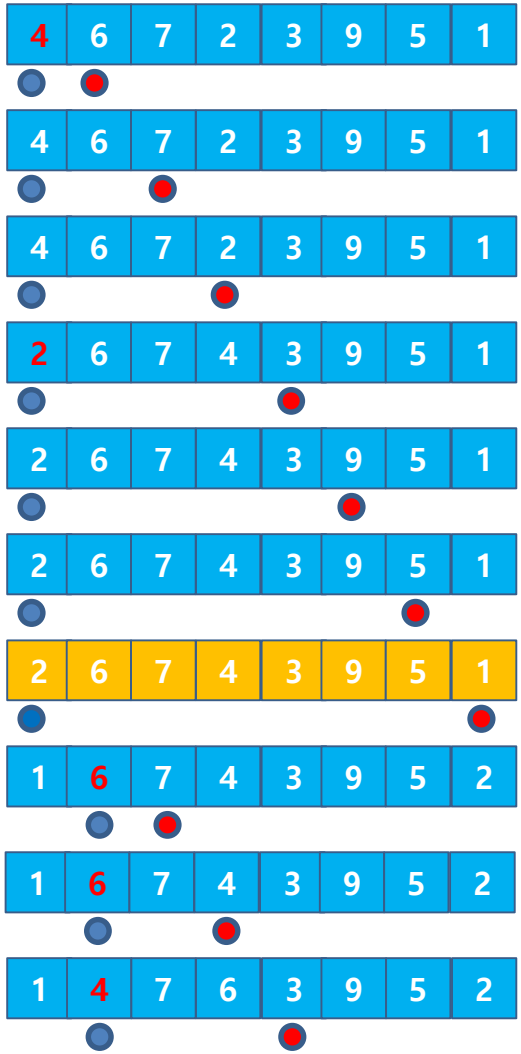




에서 까지 갈수 있는 방법 확인

- 연결이 되어 있는지?
- 연결된 다른 곳으로 이동이 가능한지?
- 이미 확인 한 데이터 인지?
- 더이상 연결된 곳이 없는지?
- Target인지?
- 경로를 찾았으면 어떤 길이 더욱 빠른가?

Sample) 8개의 데이터를 오름 차순으로 정렬



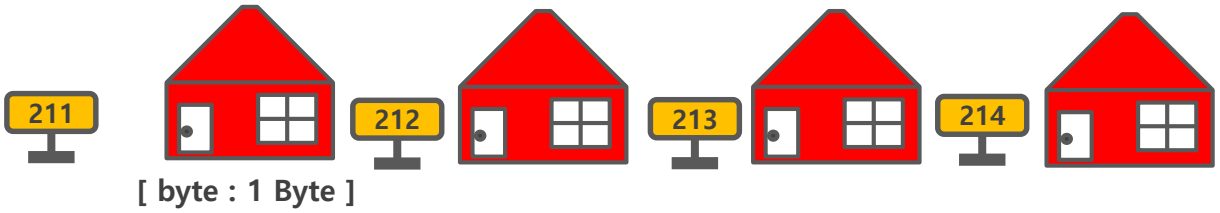

```
byte Data[] ={ 4,6,7,2,3,9,5,1 };
int  nBasicIdx = 0;

void Sort( int _checkIdx )
{
  int nCount = 0;
  byte buff = 0;

  for( nCount = (_checkIdx+1) ; nCount < 8 ; nCount++)
  {
     if( Data[nBasicIdx] > Data[nCount] )
     {
        buff = Data[nCount];
        Data[nCount] = Data[nBasicIdx];
        Data[nBasicIdx] = buff;
     }
  }

   nBasicIdx ++;
   if( nBasicIdx <=7 ) Sort(nBasicIdx);
}
```

데이터교환

무한 루프방지

* 함수 호출 : Sort(0);

Chapter 9. 배열이란?

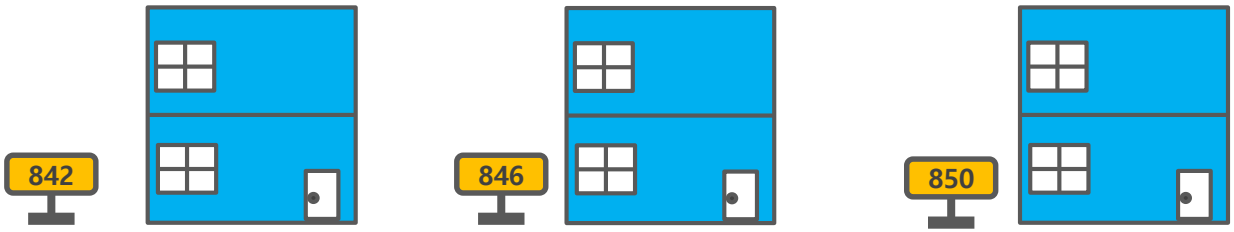


[byte : 1 Byte]

byte redHouse[4];

[int : 4 Byte]

int greenHouse[3];



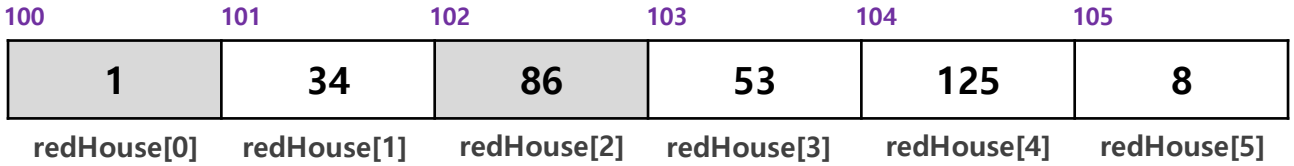
[float : 4 Byte]

float blueHouse[3];

연속된 공간(메모리)에 같은 Type의 변수를 할당하는 형태

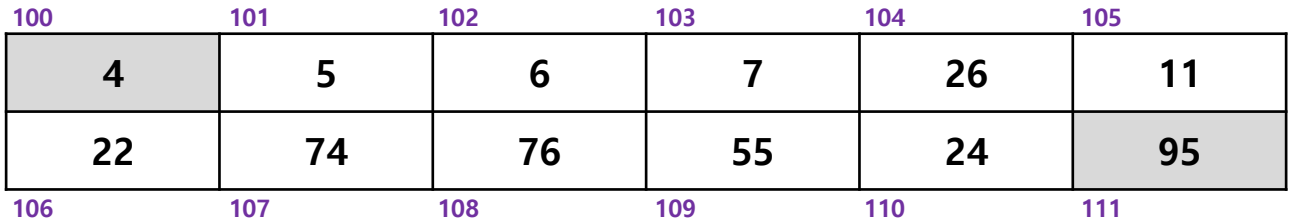
Chapter 9. 배열은 어떻게 사용하는가?

byte redHouse[6];



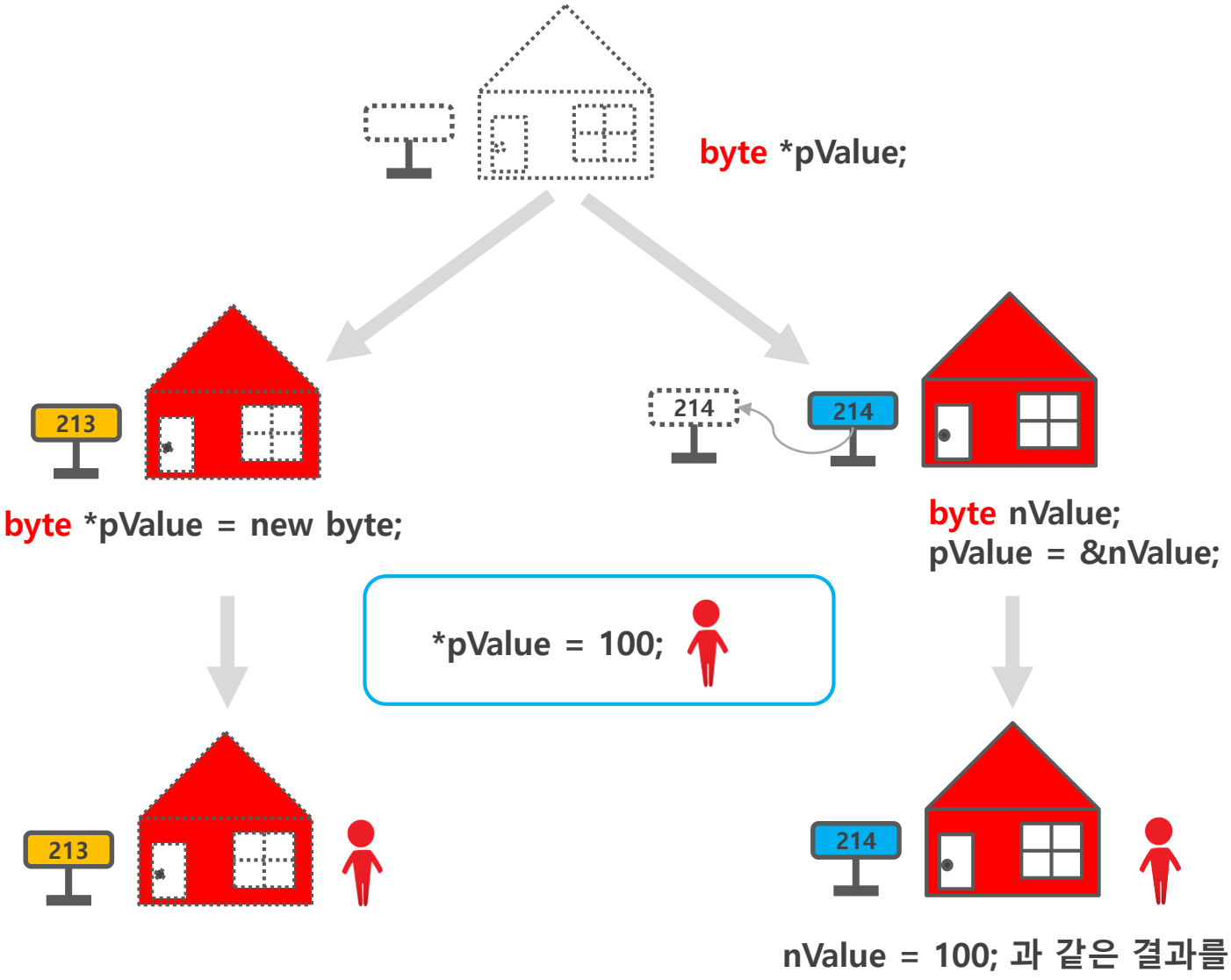
redHouse[0] → 값: 1 , 주소: 100
redHouse[2] → 값: 86 , 주소: 102
redHouse[6] → Memory Access Error

byte redHouse[2][6];



redHouse[0][0] → 값: 4 , 주소: 100
redHouse[1][5] → 값: 95 , 주소: 111

선언 할 경우에는 변수의 수량을 입력 하며 접근할 때에는 0 부터 접근한다.


byte *pValue;
213
214
214
byte *pValue = new byte;
byte nValue;
pValue = &nValue;
*pValue = 100;
213
214
nValue = 100; 과 같은 결과를 가진다.

Chapter 10. 포인터의 주소와 값을 나타내는 &(앤퍼센트) , *(아스타)

byte nValue = 5; ····▶ nValue에 값 5를 입력

pValue = &nValue; ····▶ nValue의 주소를 pValue에 연결

byte *pValue;

* 를 이용하여 포인터 변수임을 나타낸다.

pValue → 포인터 변수 pValue (주소를 저장할 수 있는 Type의 변수이다.)

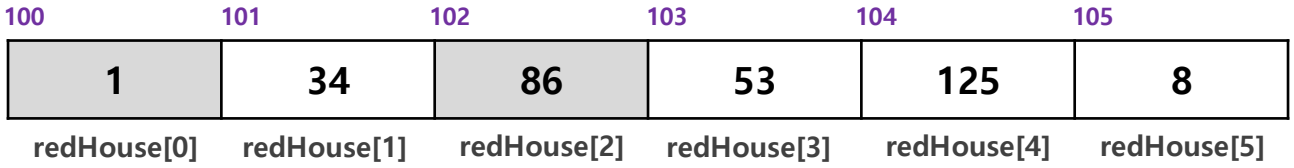
*pValue → pValue의 값

&nValue → nValue의 주소

- 포인터 변수는 new, melloc을 통해 **메모리를 할당** 하거나 다른 변수의 **주소를 연결** 시켜 주지 않으면 사용할 수 없다.
- pValue에 값을 넣을 경우

  *pValue = 10; → 실제 데이터는 nValue의 메모리에 저장된다.

byte redHouse[6];



redHouse[0] → 값: 1 , 주소: redHouse &redHouse[0] , 100

redHouse[2] → 값: 86 , 주소: &redHouse[2] , 102

byte redHouse[2][6];

100	101	102	103	104	105
4	5	6	7	26	11
22	74	76	55	24	95
106	107	108	109	110	111

redHouse[0][0] → 값: 4 , 주소: redHouse &redHouse[0][0] , 100

redHouse[1][5] → 값: 95 , 주소: &redHouse[1][5] , 111

byte *pRedHouse = new byte[6]; → delete[] pRedHouse; 메모리 해제 필요

pointer는 return , argument로 사용할 수 있다.

```
int *pValueSum = NULL;
int nValueA = 3;
int nValueB = 4;

pValueSum = SumFunction( &nValueA , &nValueB );
```

함수 호출 / 인자

```
int* SumFunction( int *_pValueA, int *_pValueB )
{
  int nSum = 0;
  nSum = *_pValueA + *_pValueB;
  return &nSum;
}
```

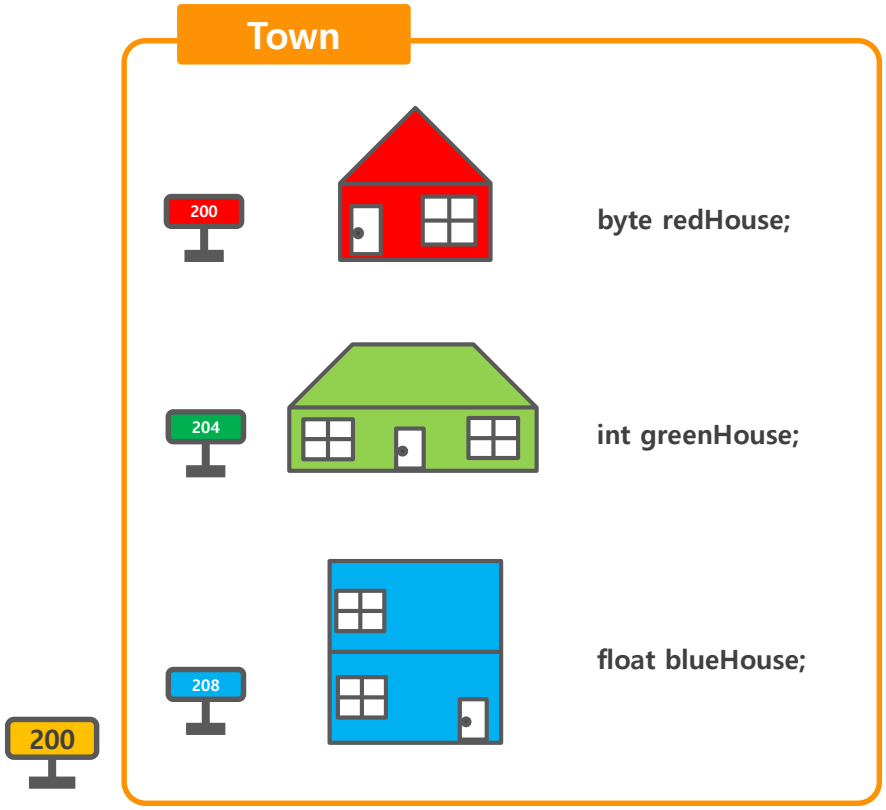
return Type / parameter / 반환 코드

int nSum; 은 지역 변수이기 때문에 return 값으로 적당하지 않다.

Chapter 13. 구조체란 무엇인가?

structure : 구조, 구조물, 체계, 짜임새

구조체는 여러 자료를 묶어서 하나의 단위로 처리하는 구조적 자료형(structured data type)이다.




```
struct town
{
    byte redHouse;
    int greenHouse;
    float blueHouse;
};
```

```
struct Human
{
    string strName;
    int nAge;
    int nSex;
    ......
};
```

```
struct Company
{
    string strName;
    int nHumanCount;
    int *pAddress;
    ........
};
```

```
struct town
{
    byte redHouse;
    int greenHouse;
    float blueHouse;
};
```



```
town stSeoul;
town stInchun[10];
town *pstKangwon;
```

```
struct _tagTown
{
    byte redHouse;
    int greenHouse;
    float blueHouse;
}stSeoul;
```

```
typedef struct _tagTown
{
    byte redHouse;
    int greenHouse;
    float blueHouse;
}stTown;
```

```
typedef struct _tagBit
{
    unsigned BIT01 : 1;
    unsigned BIT02 : 1;
    unsigned BIT03 : 1;
    unsigned BIT04 : 1;
    unsigned BIT05 : 1;
    unsigned BIT06 : 1;
    unsigned BIT07 : 1;
    unsigned BIT08 : 1;
}stBit;
```

Chapter 13. 구조체의 활용 및 주의점.

Padding 현상 : OS의 처리 단위 (32Bit, 64Bit)로 인하여 구조체 자료에 빈공간이 발생하는 현상

```
struct town
{
    byte redHouse;    →  1 Byte
    int greenHouse;   →  4 Byte
    float blueHouse;  →  4 Byte
};
```

- struct town의 크기는 실제 9 Byte 이지만 town은
  32Bit OS에서는 12 Byte, 64Bit OS에서는 16 byte로 처리된다.

```
#pragma pack(push, 1)
    // 구조체 선언
#pragma pack(pop)
```

1 Byte 단위로 메모리 할당

---

union : "조합" 이라는 뜻으로 선언된 영역의 데이터들을 한 공간의 메모리에 저장하여 사용한다.

```
typedef struct _tagBit
{
    unsigned BIT01 : 1;
    unsigned BIT02 : 1;
    unsigned BIT03 : 1;
    unsigned BIT04 : 1;
    unsigned BIT05 : 1;
    unsigned BIT06 : 1;
    unsigned BIT07 : 1;
    unsigned BIT08 : 1;
}stBit;
```

```
typedef union _tagByteData
{
    stBit stBit;
    byte Byte;
}stByteData;
```

```
stByteData.stBit.BIT01 = 1;
stByteData.stBit.BIT02 = 0;
stByteData.stBit.BIT03 = 1;
```


stByteData.Byte == 0x03;

Chapter 14. 데이터의 입/출력의 종류

[File Data]

- TEXT, PDF, Image, DB .....

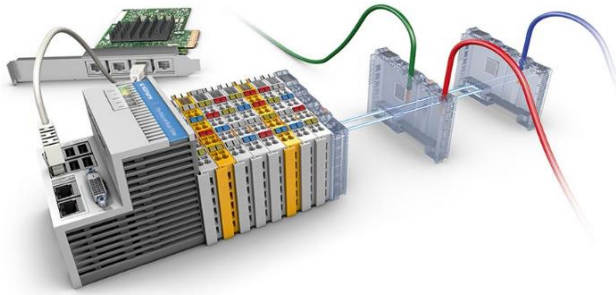
[Communication]

- Ethernet, Serial, Parallel ....

[Screen]

- Web, Game, Editor .....
  [User Interface]

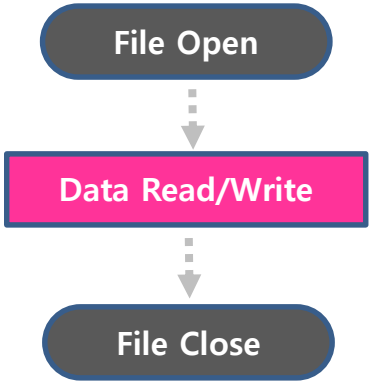
[Device Control]



- Twincat, DeviceNet, Melsec ....

[File Data]

- TEXT, PDF, Image, DB .....



Stream: 데이터의 흐름, 또는 연결 통로

→ 어떤 데이터를 처리할 것인가?

→ 어떤 방법으로 File에 데이터를 읽거나 기록 할 것인가?

Path: 경로

→ 어느 곳에 위치한 파일을 읽거나 기록 할 것인가?

Property: 속성

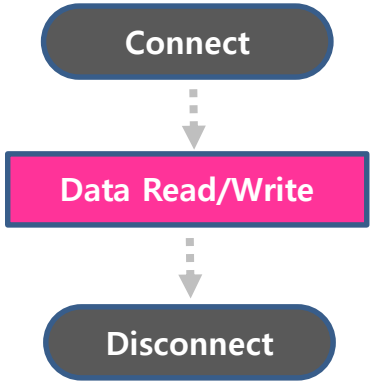
→ 파일을 읽기만 할 것인가? 쓰기만 할 것인가? 모두 할 것인가?

→ 파일을 숨길 것인가? 보일 것인가?

Chapter 14. Communication, Device Control의 데이터의 입/출력

[Communication]

- **Ethernet, Serial, Parallel ....**



Stream: 데이터의 흐름, 또는 연결 통로

→ 어떤 통신 방법을 사용할 것인가?

→ 어떤 방법으로 통신 데이터를 보내거나 받을 것인가?

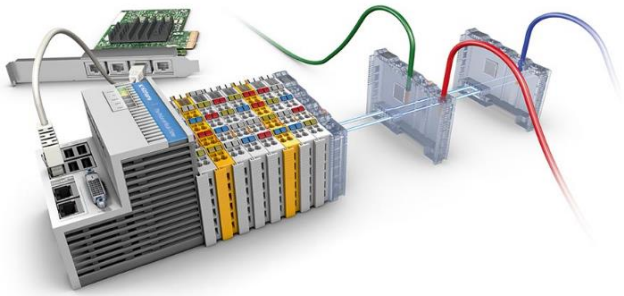
Path: 경로

→ 어떻게 연결된 통신 방법을 이용하여 데이터를 보내거나 받을 것인가?

Property: 속성

→ 선택한 통신은 어떤 형태로 데이터를 보내야 하는가? (Protocol)

[Device Control]



- **Twincat, DeviceNet, Melsec ....**

#include<stdio.h>

Stand**d**ard **I**nput **O**utput

- 표준 입출력, 즉 키보드로 입력, 모니터로 출력하는 도구(함수)를 가지고 있다. [printf(), scanf() ...]

선행 처리 표시 : Compile할 때 Main Source를 기계언어로 번역하기 전에 수행되는 부분

include 파일의 포함 : 어떠한 파일을 source에 포함하고자 할 때 정해진 Code

< > 정해진 위치에 파일이 포함되어 있음을 알림 : 프로젝트 설정이나 Editor에 따라 위치가 다름

#include"C://Data/Header/stdio.h" : 특정 경로의 파일이 포함되어 있음을 알림

#include<math.h>

Mathematics

- 산술적 계산에 필요한 도구 (함수)들을 가지고 있다. [pow(), sin() ...]

헤더 파일은 프로그램 작성에 필요한 기능(함수)들을 제공한다.

Thank You

행함이 없는 믿음은 쓸모가 없다. (Faith without deeds is useless.)