

# 1장 C 언어 복습

- # 표준 입출력
- # 배열
- # 포인터
- # 배열과 포인터
- # 함수
- # const와 포인터
- # 구조체
- # 컴파일러 사용 방법

# 1. 표준 입출력

## 표준 입출력

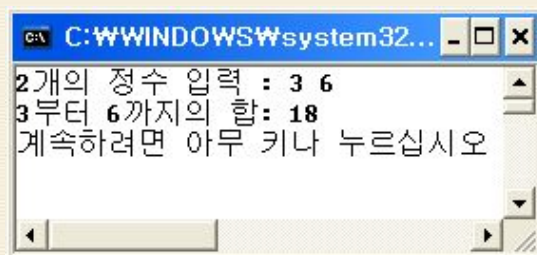
- 입력 : 키보드, scanf 함수
- 출력 : 모니터, printf 함수

문제 : 정수값 2개를 입력받고  
두 값 사이의 값들을 더하여  
출력하라.

주소 전달

서식 문자 : int형 - %d

값 전달



```
#include <stdio.h>           // 전처리문 사용
int main(void)               // main 함수
{
    int Num1, Num2;          // 변수 선언
    int Sum = 0;             // 변수 선언 및 초기화
    int i;

    printf("2개의 정수 입력 : "); // 출력
    scanf("%d %d", &Num1, &Num2); // 입력

    for (i = Num1; i <= Num2; i++) // 제어문
        Sum += i;                // 연산자

    printf("%d부터 %d까지의 합 : %d\n",
           Num1, Num2, Sum);

    return 0;
}
```

## 2. 배열

✦ 배열 : 같은 타입의 변수 여러 개를 묶어서 처리

✦ 배열 변수 선언 및 초기화의 예

```
int Grade[5];                // 지역 변수의 경우 쓰레기값을 가짐
Grade[2] = 100;              // 세 번째 원소의 값을 100으로 변경

int Grade[5] = { 10, 20, 30, 40, 50 };

int Grade[5] = { 10, 20 };    // Grade[2] 이후로는 0 값을 가짐

int Grade[5] = { 0 };        // 모두 0 값을 가짐

int Grade[2][3] = { { 10, 20, 30 }, { 40, 50, 60 } }; // 2차원 배열
Grade[1][1] = 100;           // 2행 2열 원소의 값을 100으로 변경
```

## 2. 배열

- ✦ 문제 : 5행 5열의 int형 배열을 선언하고 실행 결과와 같이 각 원소의 값을 채운 후 출력하기

```
int main(void)
{
    int Ary[5][5];                // 2차원 배열
    int i, j;

    for (i = 0; i < 5; i++) {
        for (j = 0; j < 5; j++) {
            if (i >= j)            // 왼쪽 아래 부분 : i + 1
                Ary[i][j] = i + 1;
            else                   // 오른쪽 위 부분 : 0
                Ary[i][j] = 0;
        }
    }

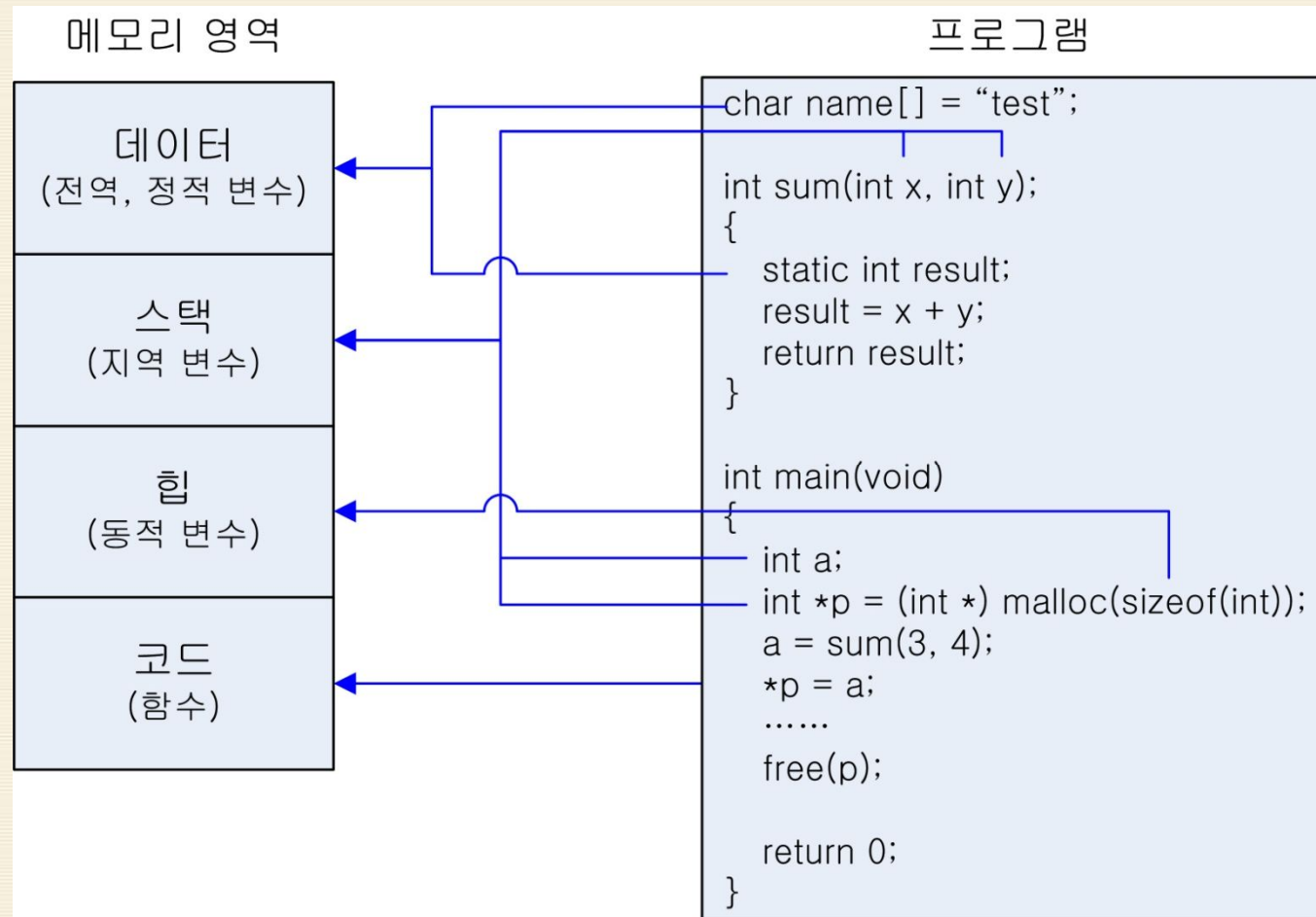
    for (i = 0; i < 5; i++) {
        for (j = 0; j < 5; j++) {
            printf("%d ", Ary[i][j]);
        }
        printf("\n");
    }

    return 0;
}
```



### 3. 포인터 : 프로그램과 메모리

#### # 실행 중인 프로그램(프로세스)의 메모리 구조



### 3. 포인터 : 메모리 주소

#### ▣ 메모리 주소

- 주소 : 바이트 단위의 번호
- 주소를 8비트로 표현한다면 →  $2^8$ 개(0~255번지)의 주소값
- Windows 운영체제에서는 32비트로 표현 →  $2^{32}$ 개(0~ $2^{32}-1$ 번지)
  - 프로세스 1개 당 4GByte의 전용 메모리(가상 메모리)가 할당됨

#### ▣ 변수와 함수의 주소

- 변수와 함수는 메모리 공간을 차지함

```
int x[5] =  
    { 0, 1, 2, 3, 4 };
```

x[0] : 1000

0

x[1] : 1004

1

x[2] : 1008

2

x[3] : 1012

3

x[4] : 1016

4

```
char y[3] =  
    { 'a', 'b', 'c' };
```

y[0] : 1000

'a'

y[1] : 1001

'b'

y[2] : 1002

'c'

```
int func1();  
int func2();
```

1000

func1

2000

func2



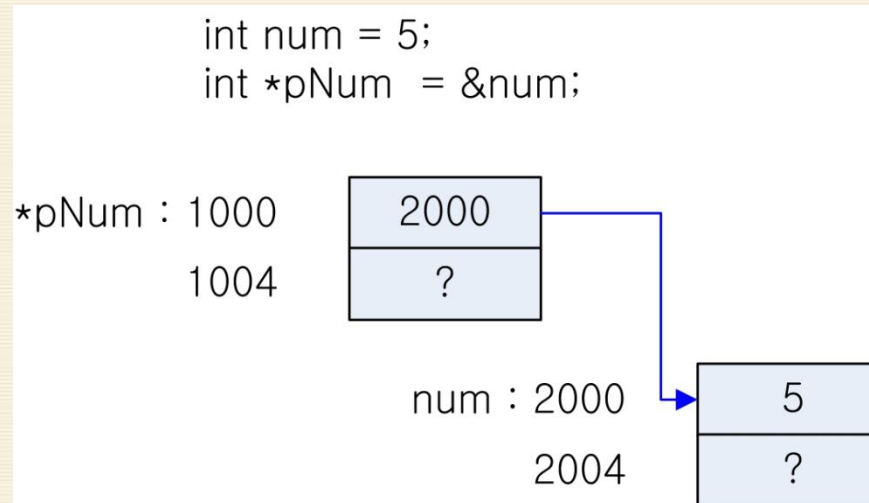
### 3. 포인터 : 변수와 포인터

#### # 변수의 주소 : 주소 연산자(&) 사용

- `int num = 5;`
- `printf("변수 num의 주소 : %d\\n", &num);`

#### # 포인터 : 주소값을 저장하는 변수

- 타입 별로 다르게 표현됨 → 포인터 변수를 통해 간접 처리 가능
- 예 : int형 포인터 변수



### 3. 포인터 : 변수와 포인터

#### ▣ 포인터 변수를 통한 변수 접근 : 역참조 연산자(\*) 사용

- `int *pNum = &num; *pNum = 5; (num = 5;와 동일)`

#### ▣ 포인터 연산 : 증가(++), 감소(--), 덧셈(+), 뺄셈(-) 가능

- `int`형 포인터의 경우 1 증가 : `int`형 변수의 크기인 4만큼 증가함

#### ▣ 예제

```
int main(void)
{
    int var = 5;
    int *p = &var;    // 포인터 변수 선언 및 var 주소로 초기화

    printf("var의주소 : %d\n", &var);
    printf("var의값   : %d\n", var);
    printf("p의주소   : %d\n", &p);
    printf("p의값     : %d\n", p);
    printf("p가가리키는변수의값 : %d\n", *p);

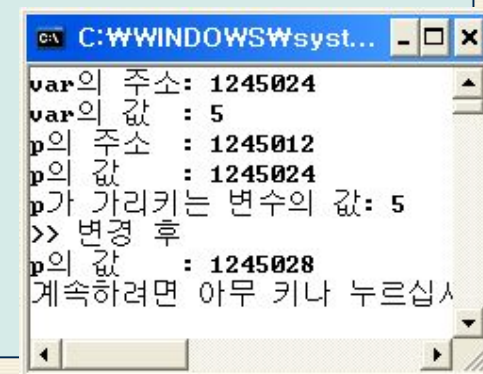
    p++;

    printf(">> 변경후\n");
    printf("p의값     : %d\n", p);

    return 0;
}
```

`int`형이므로  
4 증가

→ `p++;`

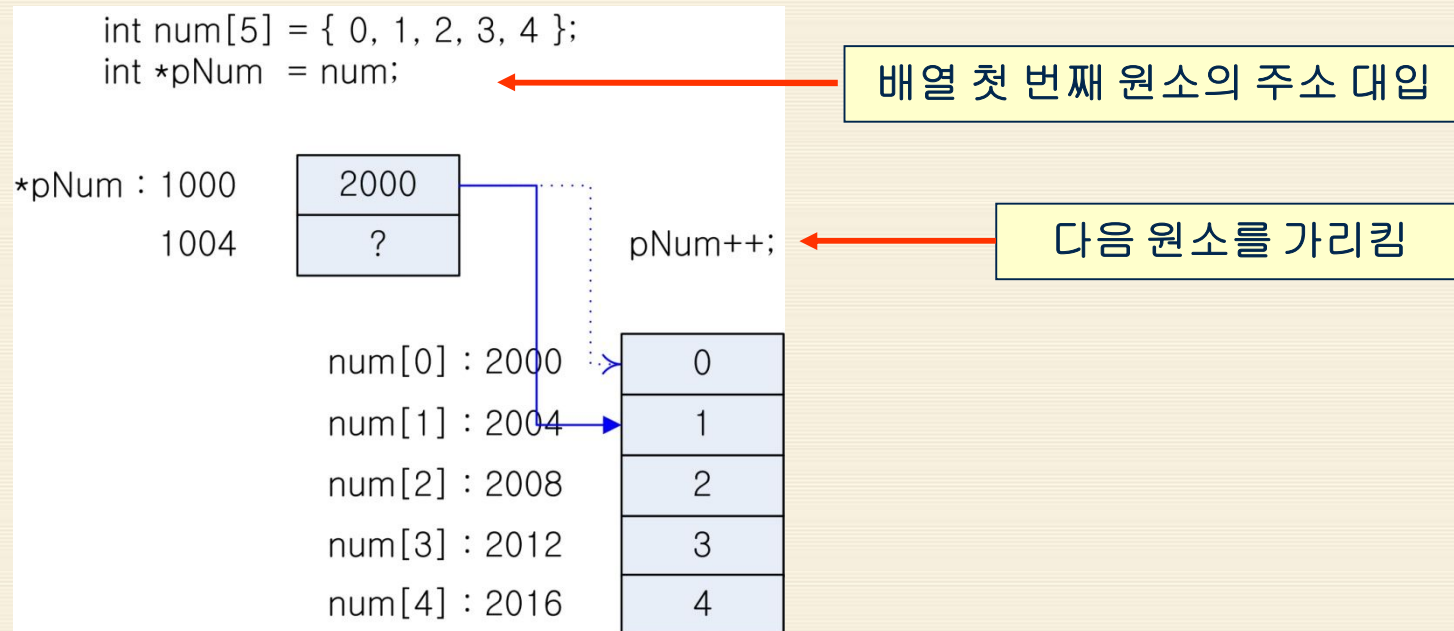


```
C:\WINDOWS\system32\cmd.exe
var의 주소: 1245024
var의 값  : 5
p의 주소  : 1245012
p의 값    : 1245024
p가 가리키는 변수의 값: 5
>> 변경 후
p의 값    : 1245028
계속하려면 아무 키나 누르십시오 . . .
```



## 4. 배열과 포인터

- # 배열은 포인터처럼, 포인터는 배열처럼 사용 가능!
- # 배열에 대한 포인터 연산의 적용 예



- pNum[0], pNum[1], ... 과 같이 사용 가능
- pNum[2] == \*(pNum + 2)

# 주의 : 배열명에 대한 주소 변경 불가 → 상수 개념



## 5. 함수

### # 함수 작성 시 고려 사항

- 함수 원형, 함수 정의, 함수 호출, 매개변수 전달

### # 예제 : $x^y$ 을 계산하는 함수

값에 의한 전달

```
#include <stdio.h>

int power(int x, int y)
{
    int i;
    int result = 1;

    for (i = 0; i < y; i++)
        result *= x;

    return result;
}
```

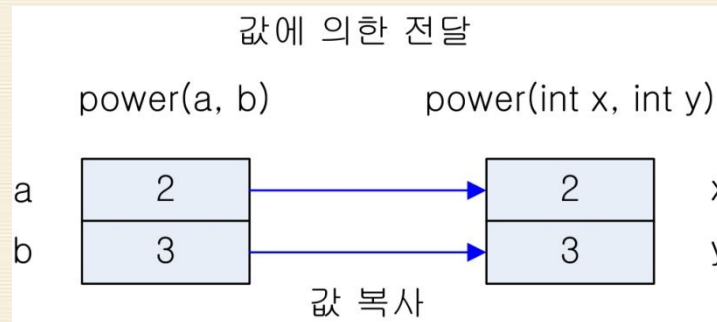
```
int main(void)
{
    int a = 2;
    int b = 3;
    int result = power(a, b); // 함수호출
    printf("%d^%d : %d\n", a, b, result);

    a = 3;
    b = 4;
    result = power(a, b);
    printf("%d^%d : %d\n", a, b, result);

    return 0;
}
```

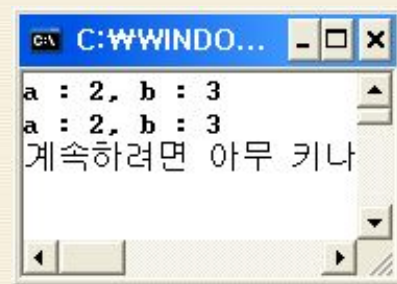
## 5. 함수 : 매개변수 전달 방식

### # 값에 의한 전달



### # 예제 : swap 함수

#### ■ 문제점은?



```
void swap(int x, int y) ← 값에 의한 전달
{
    int temp = x;
    x = y;
    y = temp;
}

int main(void)
{
    int a = 2;
    int b = 3;

    printf("a : %d, b : %d\n", a, b);

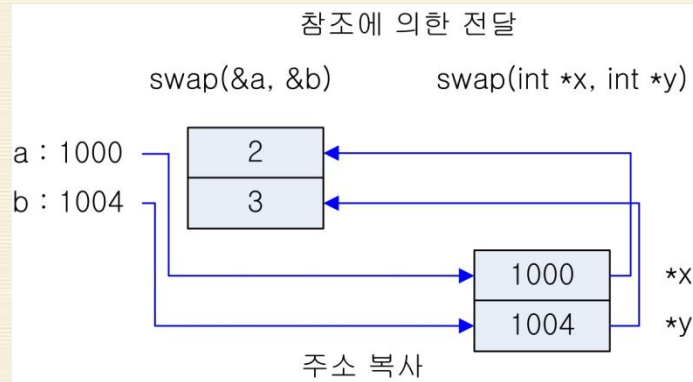
    swap(a, b);

    printf("a : %d, b : %d\n", a, b);

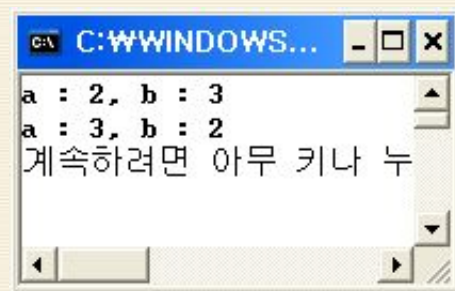
    return 0;
}
```

## 5. 함수 : 매개변수 전달 방식

### 참조에 의한 전달



### 예제 : swap 수정



```
void swap(int *x, int *y) ← 참조에 의한 전달
{
    int temp = *x;
    *x = *y;
    *y = temp;
}

int main(void)
{
    int a = 2;
    int b = 3;

    printf("a : %d, b : %d\n", a, b);

    swap(&a, &b); ← 주소 전달
    printf("a : %d, b : %d\n", a, b);

    return 0;
}
```

## 5. 함수 : 매개변수 전달 방식 - 배열의 전달

### ▣ 배열의 전달

- 값에 의한 전달 불가
- 첫 번째 원소의 주소를 전달 → 참조에 의한 전달 사용

### ▣ 예제

int형 배열 포인터, 원소 개수

```
void SetArray(int *ary, int count)
{
    int i;

    for (i = 0; i < count; i++)
        ary[i] = i * i;
}
```

포인터를 배열처럼 사용

```
int main(void)
```

```
{
```

```
    int i;
```

```
    int Ary[5];
```

```
    SetArray(Ary, 5);
```

```
    for (i = 0; i < 5; i++)
```

```
        printf("%d : %d\n", i, Ary[i]);
```

```
    return 0;
```

```
}
```

배열의 첫 번째 원소 주소,  
원소의 개수 전달

0	:	0
1	:	1
2	:	4
3	:	9
4	:	16

계속하려면 아



## 6. const와 포인터

### # const 상수 선언

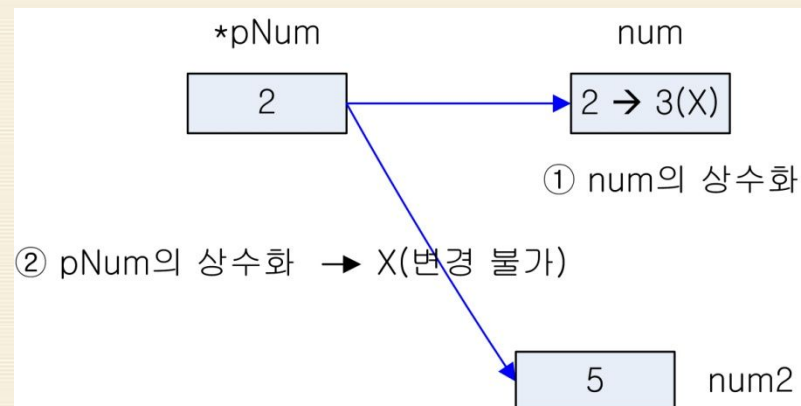
- `const double PI = 3.14;`
- 변수 PI의 값 변경 불가

### # 함수 매개변수의 상수화

- `int sum(const int x, const int y) { return (x + y); }`
- 함수 내에서 x, y의 값 변경 불가
- 실매개변수로 일반 변수와 const 상수 모두 사용 가능
  - 형식매개변수가 const가 아닌 경우에는 실매개변수로 const 상수 불가

### # const와 포인터

- `int *pNum = &num;`



## 6. const와 포인터

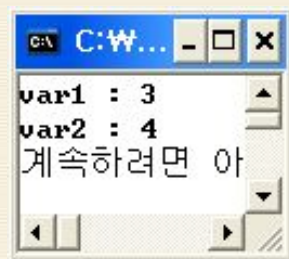
### 1. num의 상수화

- `const int *pNum = &num;`

### 2. pNum의 상수화

- `int * const pNum = &num;`

### # 예제 : const의 사용



```
int main(void)
{
    int var1 = 1;
    int var2 = 2;

    const int *p1;
    //int * const p2;
    int * const p2 = &var1;

    p1 = &var1;
    p1 = &var2;
    // *p1 = 5;
    var2 = 4;

    *p2 = 3;
    // p2 = &var2;

    printf("var1 : %d\n", var1);
    printf("var2 : %d\n", var2);

    return 0;
}
```

선언과 동시에 초기화

포인터를 통해 값 변경 불가

변수 자체는 변경 가능

다른 변수로 변경 불가

## 7. 구조체

### # 구조체

- 동질 또는 이질적인 데이터 여러 개를 하나의 그룹으로 처리
- 사용자 정의형
- 예 : 2차원 평면상의 한 점

```
struct Point {           // 구조체 선언
    int x;
    int y;
};
struct Point P1, P2; // 변수 선언
P1.x = 5;             // 변수 사용
P1.y = 6;
```

- 구조체 변수 대입 ○
  - 배열은 대입 X
- 구조체 배열 가능
- 구조체의 멤버 변수로 배열 가능
  - 이 경우에도 대입 가능

P1 = P2;

struct Point Sum(struct Point P1, struct Point P2);

값에 의한 전달 가능

```
struct Point Ary[10];
struct Student {
    char name[20];
    int id;
    int score;
};
```

## 7. 구조체

### ■ 예제 : 2차원 평면상의 한 점을 위한 Point 구조체 사용

```
struct Point {  
    int x, y;  
};
```

P1 : 값에 의한 전달, P2 : 참조에 의한 전달

```
struct Point Sum(struct Point P1, struct Point *P2)  
{  
    struct Point Po;  
    Po.x = P1.x + P2->x;  
    Po.y = P1.y + P2->y;  
  
    return Po;  
}
```

구조체 포인터의 멤버는 ->로 접근  
(\*P2).x와 동일

```
int main(void)  
{  
    struct Point A = { 1, 2 };  
    struct Point B = { 3, 4 };  
    struct Point C;
```

구조체 변수의 선언 및 초기화

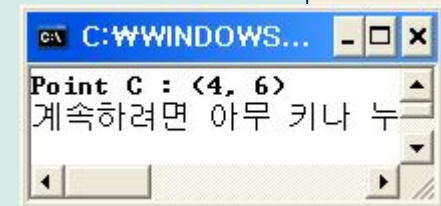
```
    C = Sum(A, &B);
```

A : 값 전달, B : 주소 전달

```
    printf("Point C : (%d, %d)\n", C.x, C.y);
```

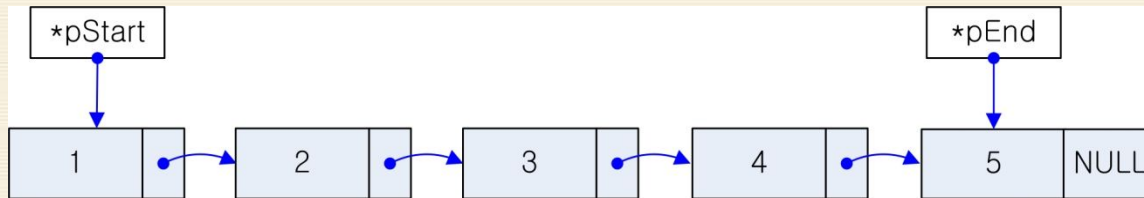
```
    return 0;
```

```
}
```



## 7. 구조체 : 단방향 링크드 리스트

### # 예제 : 단방향 링크드 리스트 만들기



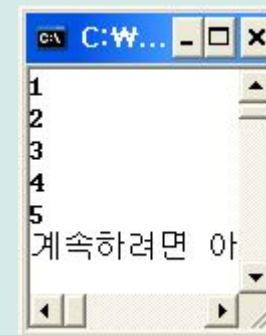
```
#include <stdio.h>
#include <stdlib.h>

struct Node {
    int data;
    struct Node *next;
};

int main(void)
{
    int i;
    struct Node *pStart = NULL;    // 첫 번째 노드 포인터
    struct Node *pEnd = NULL;      // 마지막 노드 포인터
    struct Node *Current;
```

## 7. 구조체 : 단방향 링크드 리스트

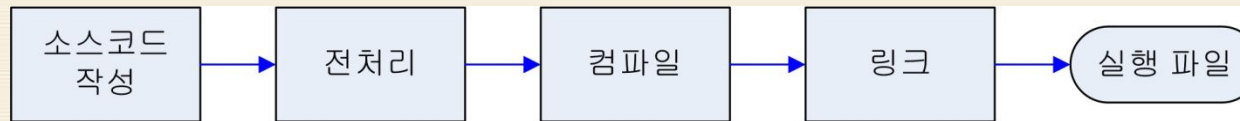
```
for (i = 1; i <= 5; i++) {  
    // 노드생성  
    Current = (struct Node *) malloc(sizeof(struct Node));  
    Current->data = i;  
    Current->next = NULL;  
  
    if (pStart == NULL)                // 첫 번째 노드  
        pStart = pEnd = Current;  
    else {                             // 노드 추가  
        pEnd->next = Current;  
        pEnd = Current;  
    }  
}  
  
// 첫 번째 노드부터 출력  
Current = pStart;  
while (Current != NULL) {  
    printf("%d\n", Current->data);  
    Current = Current->next;  
}  
  
return 0;  
}
```





## 8. 컴파일러 사용 방법

### # 프로그램 작성 및 수행 과정



### # 컴파일러의 종류

- Visual C++ : Windows 운영 체제, 통합 개발 환경 제공
  - Visual C++ 6.0 : Visual Studio 98, 아직까지도 많이 사용되고 있음
  - VC++ 8.0 : VS 2005, VC++ 9.0 : VS 2008
  - VC++ 10.0 : 최신 버전, VS 2010에 포함
- gcc : UNIX 운영 체제
- 그 외 다수

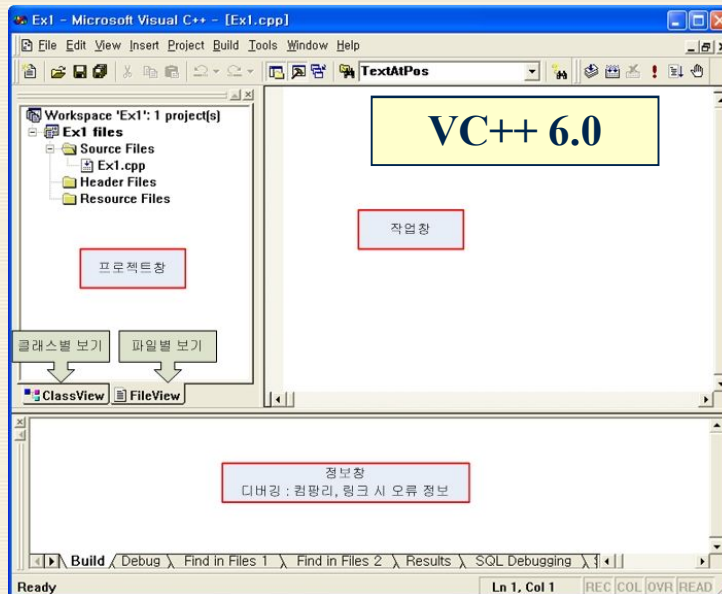
### # 컴파일러의 선택

- 어떤 컴파일러라도 상관없음
- 자신의 컴퓨터 사용 환경에 따라 적절히 선택하여 사용하면 됨
- 본 강의에서 배우는 것은 컴파일러의 사용 방법이 아닌 표준 C++!

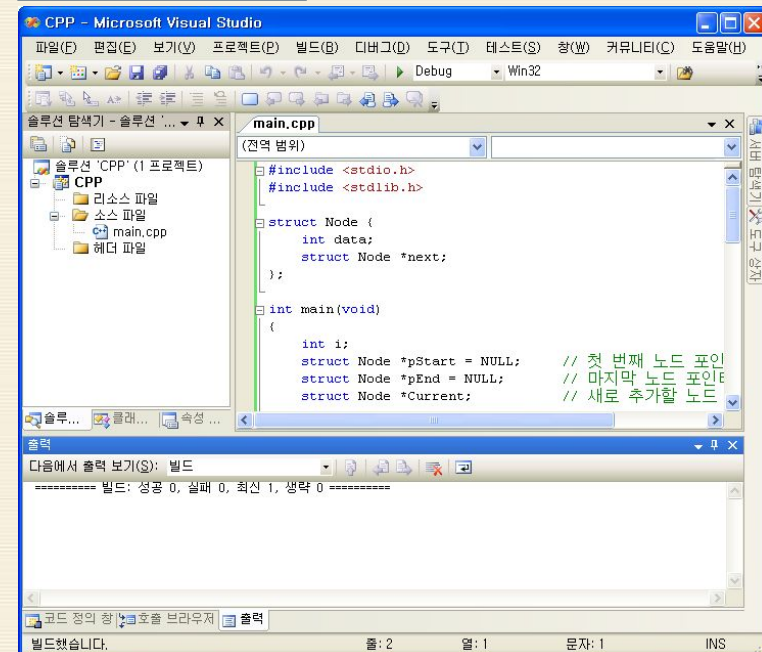
## 8. 컴파일러 사용 방법

### ✦ 본 강의에서 사용하는 컴파일러

- Visual C++ 6.0 또는 Visual C++ 8.0 이후  
→ 표준 C++ 차원(콘솔 프로그래밍)에서는 거의 유사함



VC++ 8.0



### ✦ 사용 방법 : 교재 참고

- 부록 1 : Visual C++ 6.0
- 부록 2 : Visual C++ 8.0