
Introduction to Software Design

C07. More about Array & Pointer, Command-line Arguments, Preprocessor

Tae Hyun Kim

Topics Covered

- More about Array & Pointer
 - 포인터의 배열
 - 함수 포인터 (function pointer)
 - 다차원 배열
 - 포인터의 포인터 (이중 포인터)
- Command-line Arguments
- 선행처리기 (preprocessor)

More about Array & Pointer

포인터 배열

- 포인터 변수를 여러 개 모아놓은 배열
- `int* arr[3];` // `int*`형 변수 3개를 저장하는 배열
 - 참고) `int arr[3];` // `int`형 변수 3개를 저장하는 배열

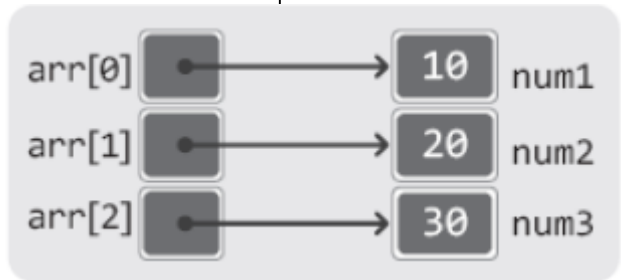
C Example

```
#include <stdio.h>

int main()
{
    int i;
    int num1=10, num2=20, num3=30;
    int* arr[3] = {&num1, &num2, &num3};

    for(i=0; i<3; ++i)
        printf("%p %d\n", arr[i], *arr[i]);

    return 0;
}
```



포인터 배열로 문자열도 여러 개 저장할 수 있다!

- `const char*` strArr[3]; // 이런 식으로..
- 왜? 문자열 하나는 아래처럼 `const char*`형 변수로 표현될 수 있으니까.
- `const char*` str1 = “string”;
- **`const char*`형 배열은 문자열(`const char*`형)을 여러 개 저장할 수 있는 배열이다.**
- “문자열 배열”이라고도 불림.

문자열 배열의 초기화

- 문자열의 초기화

```
const char* str = "aaa";
```

- 배열의 초기화

```
int arr1[5]={1, 2, 3, 4, 5};
```

- 문자열 배열의 초기화

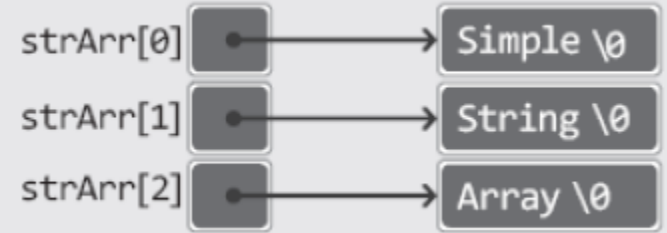
```
const char* strings[4] = {  
    "aaa",  
    "bbb",  
    "ccc",  
    "ddd"  
};
```

C & Python Examples

- C

```
#include <stdio.h>
int main()
{
    char* strArr[3] = {"Simple", "String", "Array"};
    for(int i=0; i<3; ++i)
        printf("%s\n", strArr[i]);
    return 0;
}
```

앞의 int*형 배열과 구조적으로 차이가 없다! – 모두 동일하게 각 요소가 각기 다른 메모리 공간을 가리킨다.



- Python

```
strArr = ['Simple', 'String', 'Array']

for i in range(len(strArr)):
    print(strArr[i])

for s in strArr:
    print(s)
```


함수 포인터

- 함수의 컴파일 된 바이너리 코드도 메모리 공간에 저장되어 실행된다.
- C에서 **함수의 이름**은 **함수가 저장된 메모리 공간의 주소 값**을 의미한다.
- 이러한 함수의 주소 값을 포인터 변수에 저장할 수 있다 : 함수 포인터
- 함수 포인터 변수를 선언하려면 함수 포인터의 형 (type)을 알아야 한다.

함수 포인터의 형(type)

- → 반환형과 매개변수 선언에 의해 결정된다.

- `int add(int n1, int n2)`

- → 반환형 `int`, 매개변수 `int, int`

- `void scale2x(Point* pp)`

- → 반환형 `void`, 매개변수 `Point*`

이것이 함수
포인터의 형이다.

함수 포인터의 선언

- 반환형 `int`, 매개변수 `int, int` 인 함수의 주소를 저장할 수 있는 포인터 변수 `fptr`의 선언
- `int (*fptr)(int, int);`
- 위의 `fptr` 변수에 `add` 함수의 주소값을 대입할 수 있다.
- `fptr = add;`
- 함수 포인터를 통해 함수를 호출할 수도 있다.
- `int num3 = fptr(num1, num2);`

C & Python Examples

- C

```
#include <stdio.h>
int add(int n1, int n2)
{
    return n1+n2;
}
int sub(int n1, int n2)
{
    return n1-n2;
}
int main()
{
    int (*fptr)(int, int);

    fptr = add;
    printf("%d\n", fptr(3, 5));

    fptr = sub;
    printf("%d\n", fptr(3, 5));
    return 0;
}
```

- Python

```
def add(n1, n2):
    return n1 + n2

def sub(n1, n2):
    return n1 - n2

fobj = add
print(fobj(3, 5))

fobj = sub
print(fobj(3, 5))
```

- Python은 함수도 일종의 객체(object)이다.
- 동적으로 변수의 타입이 결정되기 때문에 자유롭게 임의의 변수에 임의의 함수를 대입할 수 있다.

2차원 배열

- `int arr1d[10];`
 - `// int형 데이터를 10개 가지는 배열 (1차원)`
- `int arr2d[3][4];`
 - `// 행의 개수 3, 열의 개수 4인 2차원 int형 배열`
 - `// (실제로는 [길이 4인 int형 배열]을 3개 가지는 배열이다)`
- `TYPE arr[행의 개수][열의 개수];`

2차원 배열 요소의 접근

- `arr1d[index] = 10; // 1차원 배열에서 각 요소에 접근`
- `arr2d[행 방향 index][열 방향 index] = 10;`
- `// 2차원 배열은 이렇게 각 요소에 접근할 수 있다.`

	1열	2열	3열	4열
1행	[0][0]	[0][1]	[0][2]	[0][3]
2행	[1][0]	[1][1]	[1][2]	[1][3]
3행	[2][0]	[2][1]	[2][2]	[2][3]

`int arr1[3][4];`

	1열	2열	3열	4열	5열	6열
1행	[0][0]	[0][1]	[0][2]	[0][3]	[0][4]	[0][5]
2행	[1][0]	[1][1]	[1][2]	[1][3]	[1][4]	[1][5]

`int arr2[2][6];`

2차원 배열 요소의 접근 예

```
int arr[3][3];
```



배열 생성

	1열	2열	3열
1행	0	0	0
2행	0	0	0
3행	0	0	0

(모든 배열 요소가 0으로 초기화 된 상태라고 가정)

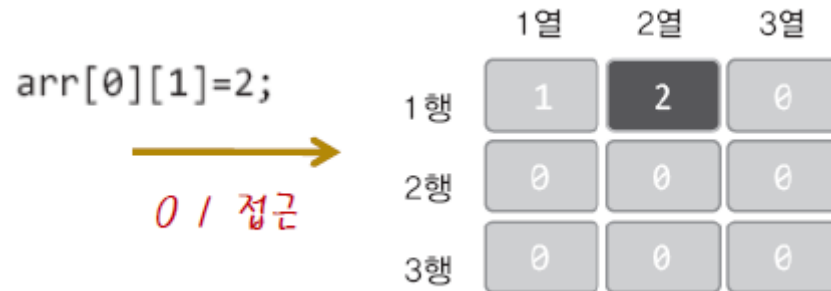
```
arr[0][0]=1;
```



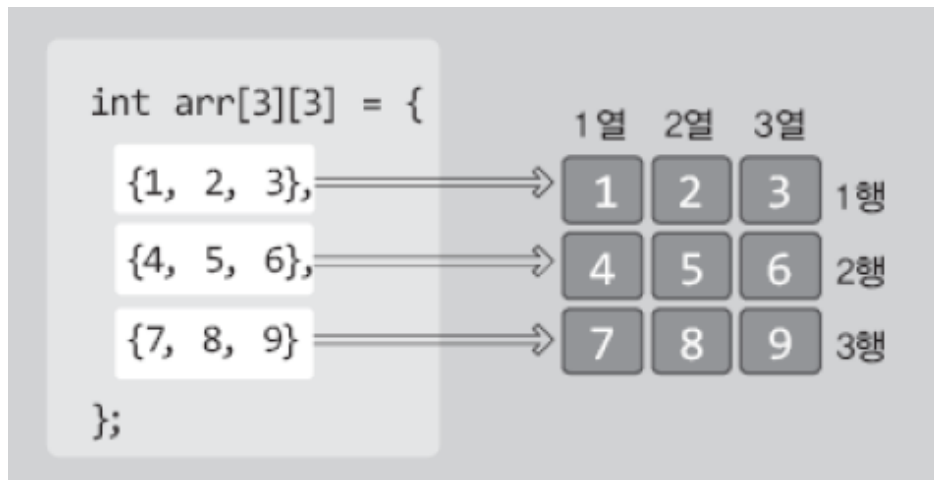
0 0 접근

	1열	2열	3열
1행	1	0	0
2행	0	0	0
3행	0	0	0

2차원 배열 요소의 접근 예



2차원 배열을 선언과 동시에 초기화



초기화 리스트 안에
여러 개의 초기화
리스트가 들어간다.

(구조체 배열의
초기화와 비슷)

메모리 공간은 1차원이라고 하지 않았나요?

- 그런데 어떻게 “2차원 배열”을 저장하죠?
- `int arr[2][3]`; 배열의 각 요소 주소를 출력해보면...

	1열	2열	3열
1행	55032 40	55032 44	55032 48
2행	55032 52	55032 56	55032 60

- 2차원 배열도 메모리상에는 1차원의 형태로 저장된다.

40	41	42	43	44	48	46	47	48	49	50	51	52	53	54	55	56	57	58	59	60	61	62	63
arr[0][0]				arr[0][1]				arr[0][2]				arr[1][0]				arr[1][1]				arr[1][2]			
arr[0]												arr[1]											

→ `int arr[2][3]` : [길이 3인 `int`형 배열]을 2개 가지는 배열

C & Python Examples

- C

```
#include <stdio.h>
int main()
{
    int arr2d[2][3] = {
        {1,2,3},
        {4,5,6}
    };

    for(int row=0; row<2; row++)
    {
        printf("row[%d] ", row);
        for(int col=0; col<3; col++)
            printf("%d ",
arr2d[row][col]);
        printf("\n");
    }
    return 0;
}
```

- Python

```
arr2d = [[1,2,3],
         [4,5,6]]

for row in range(2):
    print('row[%d] ' % row, end='')
    for col in range(3):
        print('%d ' % arr2d[row][col],
end='')
    print()
```

(참고) 3차원 배열

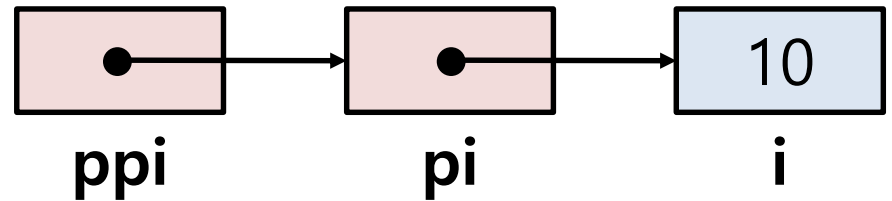
- `int arr1d[10];`
 - // 길이가 10인 1차원 int형 배열
- `int arr2d[3][4];`
 - // 행의 개수 3, 열의 개수 4인 2차원 int형 배열
- `int arr3d[3][4][5];`
 - // 행의 개수 3, 열의 개수 4, 깊이방향 개수 5인 3차원 int형 배열

포인터의 포인터?

- `int* pi;`
- **int형** 변수의 주소를 저장할 수 있는 포인터
- `int** ppi;`
- **int*형** 변수(포인터)의 주소를 저장하는 포인터
- 포인터의 포인터 혹은 이중 포인터라 부른다.

이중 포인터

```
int i = 10;  
int* pi = &i;  
int** ppi = &pi;
```



- `*pi` : 변수 `i`를 의미함
 - `*ppi` : 변수 `pi`를 의미함
 - `**ppi` : 변수 `i`를 의미함
-
- 하지만 `&&`같은 연산자는 없다.

C Example

```
#include <stdio.h>

int main()
{
    int i = 10;
    int* pi = &i;
    int** ppi = &pi;

    // 아래 세 문장은 같은 일을 한다.
    i = 20;
    *pi = 20;
    **ppi = 20;

    // 아래 세 문장은 같은 일을 한다.
    printf("%d\n", i);
    printf("%d\n", *pi);
    printf("%d\n", **ppi);

    return 0;
}
```

이중 포인터는 어디에 쓸까? - 1

- 문자열을 swap 하는 함수!
 - `const char* str1 = “aaa”; const char* str2 = “bbb”;`
 - `str1`이 가리키는 문자열과 `str2`가 가리키는 문자열을 바꾸는 함수?
 - 다시 말하면 `str1`에 저장되어 있는 주소값과 `str2`에 저장되어 있는 주소값을 바꾸는 함수
- 포인터 예
 - `int`형 변수 2개의 값을 서로 바꾸는 함수
 - `void swap(int* pi1, int* pi2)`
- 이중 포인터 예
 - `const char*`형 변수 2개의 값을 서로 바꾸는 함수
 - `void swap(const char** pstr1, const char** pstr2)`

C Example

```
#include <stdio.h>

void swap(const char** pstr1,
const char** pstr2)
{
    const char* temp = *pstr1;
    *pstr1 = *pstr2;
    *pstr2 = temp;
}

int main()
{
    const char* str1 = "aaa";
    const char* str2 = "bbb";

    printf("%s %s\n", str1, str2);

    swap(&str1, &str2);

    printf("%s %s\n", str1, str2);

    return 0;
}
```

예전에 작성했던 int형 변수값을 바꾸는 swap함수와 비교해보자

```
#include <stdio.h>

void swap(int* p1, int* p2)
{
    int temp = *p1;
    *p1 = *p2;
    *p2 = temp;
}

int main()
{
    int num1=10, num2=20;
    swap(&num1, &num2);
    printf("%d %d\n", num1,
num2);
    return 0;
}
```

이중 포인터는 어디에 쓸까? - 2

- 포인터 배열을 함수의 인자로 넘길 때
 - `const char* strArr[] = {"aaa", "bbb", "ccc"};`
 - 이것을 함수의 인자로 어떻게 넘겨야 할까?
- 포인터 예
 - `int`형 배열의 내용을 출력하는 함수
 - `void printArray(int* arr, int len)`
- 이중 포인터 예
 - `const char*`형 배열의 내용을 출력하는 함수
 - `void printArray(const char** strArr, int len)`

C Example

```
#include <stdio.h>

void printArray(const char**
arr, int len)
{
    printf("Array ");
    for(int i=0; i<len; i++)
        printf("[%d]:%s, ", i,
arr[i]);
    printf("\n");
}

int main()
{
    const char* strArr[] =
{"aaa", "bbb", "ccc"};
    printArray(strArr,
sizeof(strArr)/sizeof(char*));

    return 0;
}
```

예전에 작성했던 int형 배열을
출력하는 printArray함수와
비교해보자

```
#include <stdio.h>

void printArray(int* arr, int len)
{
    printf("Array ");
    for(int i=0; i<len; i++)
        printf("[%d]:%d, ", i,
arr[i]);
    printf("\n");
}

int main()
{
    int arr[] = {5, 10, 15, 20,
25};
    printArray(arr,
sizeof(arr)/sizeof(int));

    return 0;
}
```

Command-line Arguments

Command-line Arguments

- 프로그램 실행 시 main함수로 전달할 인자를 나열할 수 있다.
- 아래와 같이 parameter가 있는 main함수를 만들면 된다!

```
int main(int argc, char* argv[])
```

Command-line Arguments

```
int main(int argc, char* argv[])
```

- argc: 전달된 인자의 개수 (실행파일 이름 포함)
- argv: 문자열 배열(포인터 배열)의 시작주소 (이중포인터)
 - 참고) 함수 인자로 포인터 전달 시 func(int* p) 혹은 func(int p[]) 사용 가능
- 인자는 공백을 기준으로 구분된다.
- 공백을 포함한 문자열을 하나의 인자로 입력하고 싶으면, 큰 따옴표로 묶으면 된다.

```
$ ./hello_world 1 abc 0.00 "see you later."
```

```
-> argc: 5  
    argv[0]: ./hello_world  
    argv[1]: 1  
    argv[2]: abc  
    argv[3]: 0.00  
    argv[4]: see you later.
```

C & Python Examples

- C

```
#include <stdio.h>

int main(int argc, char* argv[])
{
    printf("argc: %d\n", argc);

    for (int i = 0; i < argc; ++i)
        printf("argv[%d]: %s\n", i, argv[i]);

    return 0;
}
```

```
$ ./a.out aaa bb "hello world"
10 2.1
argc: 6
argv[0]: ./a.out
argv[1]: aaa
argv[2]: bb
argv[3]: hello world
argv[4]: 10
argv[5]: 2.1
```

- Python

```
import sys

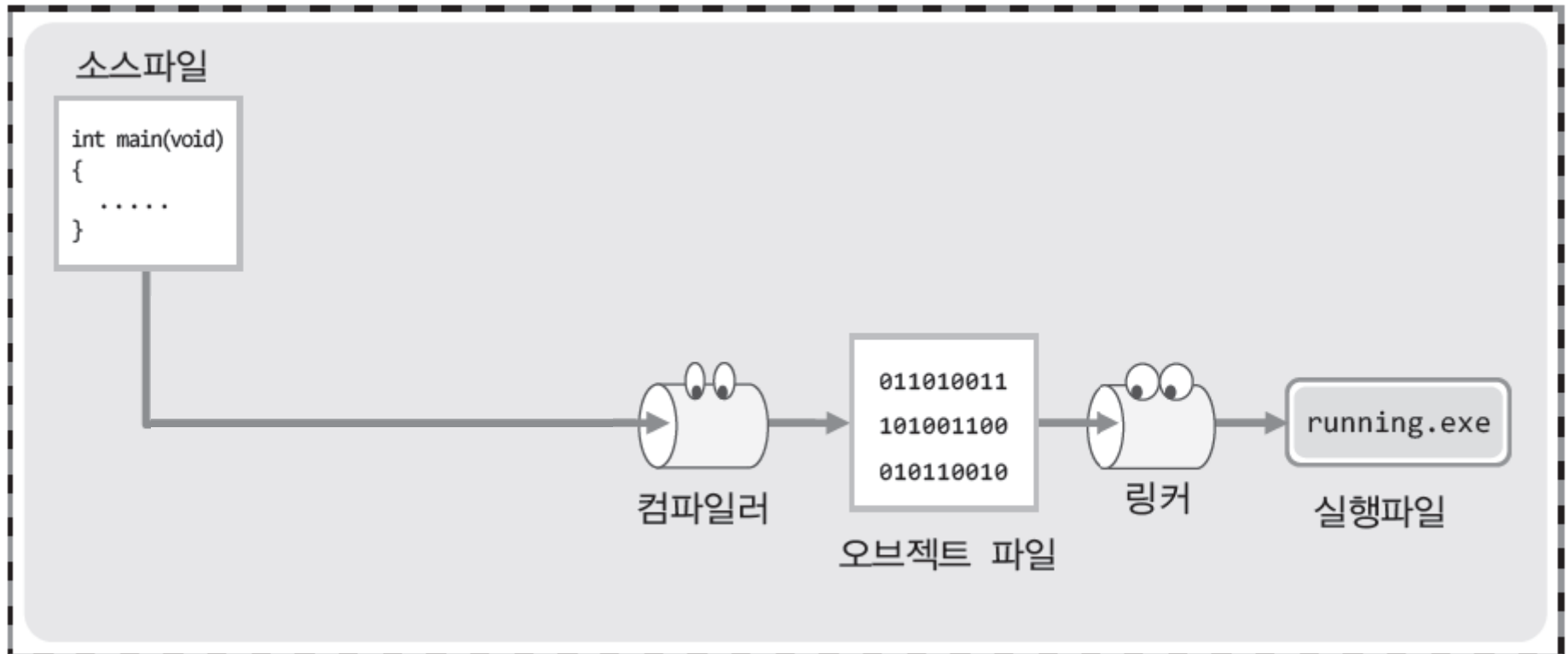
print('len(argv): %d'%len(sys.argv))

for i in range(len(sys.argv)):
    print('argv[%d]: %s'%(i, sys.argv[i]))
```

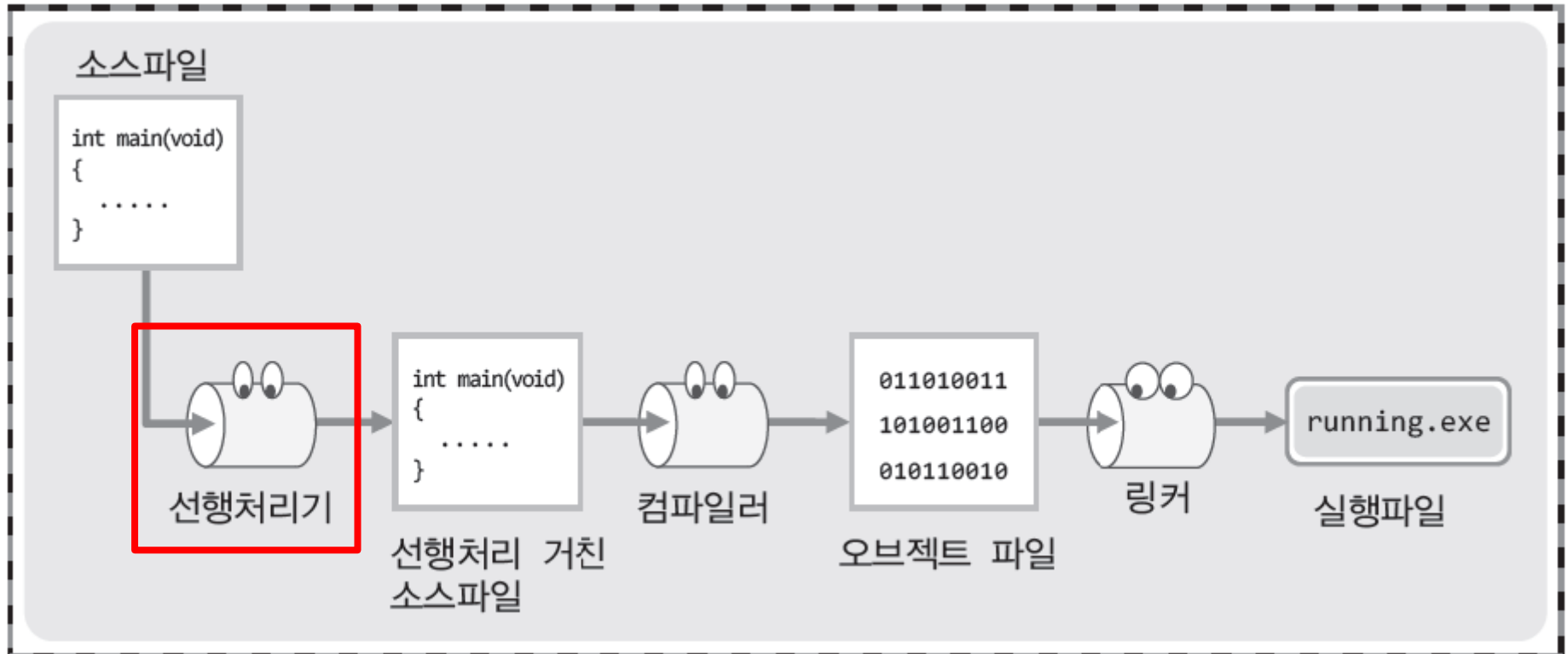
```
$ python test.py aaa bb "hello
world" 10 2.1
len(argv): 6
argv[0]: test.py
argv[1]: aaa
argv[2]: bb
argv[3]: hello world
argv[4]: 10
argv[5]: 2.1
```

Preprocessor

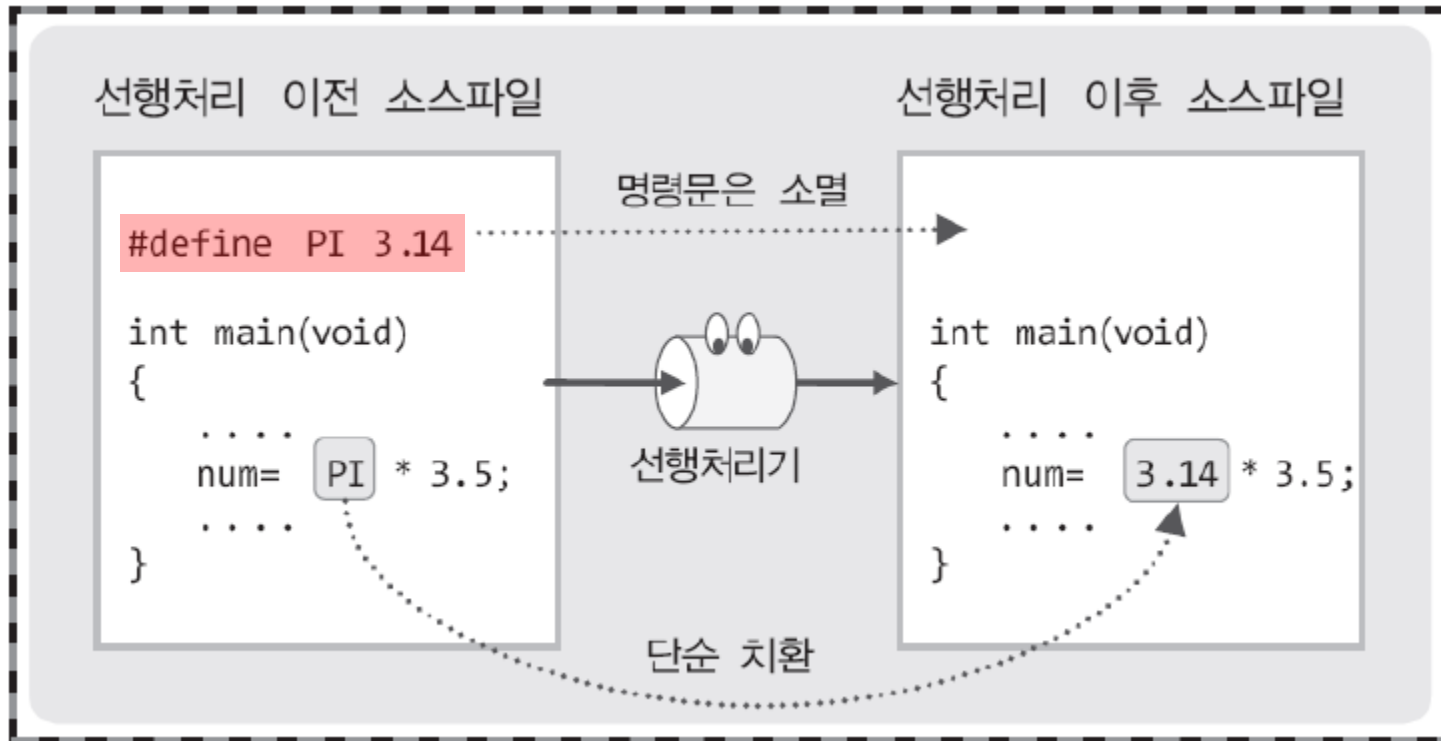
프로그램 작성 과정



선행처리기(Preprocessor, 전처리기)



간단한 예



- 선행처리 지시자 : ‘#’으로 시작한다
- #define, #include, #ifdef...

#define : 무엇인가를 ‘정의’한다

- #define **PI 3.14**
 - : **PI**를 **3.14**이라고 정의한다. (코드에서 **PI**를 **3.14**로 바꾼다.)
 - 미리 정의해놓은 상수처럼 쓸 수 있다 (매크로 상수라고 불림)
 - 관례적으로 **대문자**로 이름을 정한다.

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

```
#define NAME "John"
```

```
#define AGE 24
```

```
int main()
```

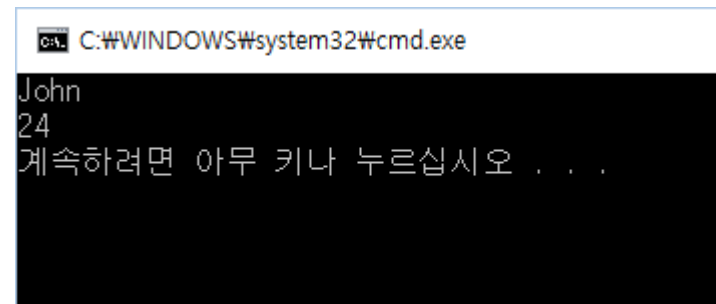
```
{
```

```
    printf("%s\n", NAME);
```

```
    printf("%d\n", AGE);
```

```
    return 0;
```

```
}
```



A screenshot of a Windows command prompt window. The title bar shows the path 'C:\WINDOWS\system32\cmd.exe'. The window content displays the output of the C program: 'John' on the first line, '24' on the second line, and '계속하려면 아무 키나 누르십시오 . . .' on the third line.

(참고) 매크로 상수와 전역 상수

- 매크로 상수는.. (예: #define [PI 3.14](#))
 - 자료형을 지정하지 않기 때문에 찾기 어려운 버그를 만들 수 있음.
 - 디버깅 시 값을 확인할 수 없기 때문에 디버깅이 까다롭다.
- #define PI 3.14
- : 그렇기 때문에 매크로 상수 대신
- `const double PI = 3.14;`
- : 전역 상수를 쓰는 것도 한 방법.
- 하지만 C에서는 여러 .c 파일에서 걸쳐 동일한 이름의 전역 상수가 정의되는 경우에는 에러가 발생하므로 사용에 제한이 있다 (하나의 .c 파일에서만 정의되는 경우에는 문제 없음).
- (참고) C++에서는 위의 사용상 제한이 없기 때문에 매크로 상수 대신 전역 상수를 사용하는 것이 권장된다.

C & Python Examples

- C

```
#include <stdio.h>

#define PI 3.14
#define NAME "John"
#define AGE 24

int main()
{
    printf("%f\n", PI);
    printf("%s\n", NAME);
    printf("%d\n", AGE);
    return 0;
}
```

- Python

```
PI = 3.14
NAME = 'John'
AGE = 24

print('%f'%PI)
print('%s'%NAME)
print('%d'%AGE)
```

- Python은 compiled language가 아니기 때문에 preprocessor의 개념 자체가 없다.
- Python에는 상수(const)의 개념도 없다.
- 프로그래머가 변수를 만든 후 값을 변경하지 않고 쓰면 된다.

#define을 함수처럼 쓰는 법

- #define SQUARE(x) ((x)*(x))
 - : 코드에서 SQUARE(x)를 ((x)*(x))로 바꾼다. (x는 함수의 인자처럼 쓰인다.) – 매크로 함수라고 불림
- #define SQUARE(x) x*x - 만일 이렇게 쓰면?

```
int num = SQUARE(3+2);
```



```
int num = 3+2*3+2; // 11
```

```
int num = 120 / SQUARE(2);
```



```
int num = 120 / 2 * 2; // 120
```

매크로 함수의 사용은 권장하지 않음

- 단점
 - 구현 및 수정이 번거롭고 실수를 하기 쉽다.
 - 자료형을 지정하지 않기 때문에 찾기 어려운 버그를 만들 수 있음.
 - 디버깅이 어렵다.
- 장점
 - 함수 호출의 오버헤드가 없다. (호출 및 리턴 시 점프, 인자와 반환값을 복사하는 오버헤드)
- (참고) 위의 장점은 가지지만, 단점은 없는 방법이 있다! – 인라인(inline) 함수

(참고) 인라인(inline) 함수

- 인라인 함수
 - 함수 호출 시 분리된 위치의 함수 코드로 점프하는 것이 아니라, 함수 호출 부분을 함수 전체 코드(컴파일된 코드)로 치환하여 컴파일하는 함수.
 - 치환은 전처리 단계가 아니라 컴파일 단계에서 일어남.
- gcc에서는 함수 앞에 inline이라는 키워드를 붙이면 된다.

```
inline void print99()  
{  
    ...  
}
```

C & Python Examples

- C

```
#include <stdio.h>

#define ADD(a,b) ((a)+(b))
#define SQUARE(x) ((x)*(x))

int main()
{
    double a = 3.14, b = 2.25;

    printf("%f\n", SQUARE(a));
    printf("%f\n", ADD(a, b));

    return 0;
}
```

- Python

```
def add(a, b):
    return a + b

def square(x):
    return x * x

a = 3.14
b = 2.25
print('%f'%square(a))
print('%f'%add(a, b))
```

- Python에는 macro function 혹은 inline function의 개념이 없다.

#ifdef, #else, #endif

- 조건부 컴파일 - 특정한 이름이 define 되어 있는지 여부에 따라 다른 코드가 컴파일 되도록 함.

```
#define TEST // 이렇게 하는 것도 가능. 말 그대로  
TEST라는 이름을 define만 한다는 의미.  
...  
#ifdef TEST  
    printf("test mode\n"); // 이 문장이 컴파일 & 실행됨  
#else  
    printf("production mode\n");  
#endif  
...
```

- #ifndef – 특정한 이름이 define되어 있지 않을 경우 해당 코드 컴파일

C Example

```
#include <stdio.h>

#define TEST

int main()
{
#ifdef TEST
    printf("test mode\n");
#else
    printf("production mode\n");
    printasdfsdf("production mode\n"); // 컴파일이 안
    되기 때문에 빌드 에러도 발생하지 않음
#endif

// 참고
#ifdef _WIN32
    printf("This is a windows machine.\n");
#else
    printf("This is a unix / linux / mac machine.\n");
#endif

    return 0;
}
```

- Python은 compiled language가 아니기 때문에 preprocessor의 개념 자체가 없다.
- 비슷한 것을 하려면 if, else를 이용해 runtime에 분기를 하면 된다.

Preprocessor 정리

- 매크로 상수 `#define PI 3.14`
 - 전역 상수를 써도 됨: `const double PI = 3.14;` (파일 하나에서만 쓸 때)
- 매크로 함수 `#define SQUARE(x) ((x)*(x))`
 - 쓰지 말 것!
- `#ifdef`, `#else`, `#endif` – 조건부 컴파일