

## Chapter 19. 함수 포인터와 void 포인터



## Chapter 19-1. 함수 포인터와 void 포인터

# 함수 포인터의 이해

## 1. 함수 포인터

1. **함수의 이름**은 함수가 저장된 메모리 공간을 가리키는 포인터이다(함수 포인터).
2. 함수의 이름이 의미하는 주소 값은 **함수 포인터 변수**를 선언해서 저장할 수 있다.
3. 함수 포인터 변수를 선언하려면 함수 포인터의 형(type)을 알아야 한다.

## 2. 함수 포인터의 형(type)

1. 함수 포인터의 형 정보에는 **반환형**과 **매개변수 선언**에 대한 정보를 담기로 약속
2. 즉, 함수의 반환형과 매개변수 선언이 동일한 두 함수의 함수 포인터 형은 일치한다.

## 3. 함수 포인터 형 결정

`int SimpleFunc(int num)`    반환형 **int**, 매개변수 **int형 1개**

`double ComplexFunc(double num1, double num2)`    반환형 **double**, 매개변수 **double형 2개**

# 적절한 함수 포인터 변수의 선언

```
int (*fptr) (int)
```

fptr은 포인터!

```
int (*fptr) (int)
```

반환형이 int인 함수 포인터!

```
int (*fptr) (int)
```

매개변수 선언이 int 하나인 함수 포인터!

함수 포인터 변수를 선언하는  
방법

```
int SoSimple(int num1, int num2) { . . . . }
```

```
int (*fptr) (int, int);
```

SoSimple 함수이름과 동일한 형의 변수 선언

```
fptr=SoSimple;
```

상수의 값을 변수에 저장

```
fptr(3, 4);
```

// SoSimple(3, 4)와 동일한 결과를 보임

함수 포인터 변수에 저장된 값을 통해서도 함수호출 가능!

```
void SimpleAdder(int n1, int n2)
{
    printf("%d + %d = %d \n", n1, n2, n1+n2);
}

void ShowString(char * str)
{
    printf("%s \n", str);
}

int main(void)
{
    char * str="Function Pointer";
    int num1=10, num2=20;

    void (*fptr1)(int, int) = SimpleAdder;
    void (*fptr2)(char *) = ShowString;

    /* 함수 포인터 변수에 의한 호출 */
    fptr1(num1, num2);
    fptr2(str);
    return 0;
}
```

# 함수 포인터 변수 관련 예제

---

```
void SimpleAdder(int n1, int n2)
{
    printf("%d + %d = %d \n", n1, n2, n1+n2);
}

void ShowString(char * str)
{
    printf("%s \n", str);
}

int main(void)
{
    char * str="Function Pointer";
    int num1=10, num2=20;

    void (*fptr1)(int, int) = SimpleAdder;
    void (*fptr2)(char *) = ShowString;

    /* 함수 포인터 변수에 의한 호출 */
    fptr1(num1, num2);
    fptr2(str);
    return 0;
}
```

10 + 20 = 30

Function Pointer

실행결과

교재에 있는 UsefulFunctionPointer.c를 통해서 함수 포인터 변수가 매개변수로 선언이 됨을 확인  
하기 바랍니다.

---



## 실습 2

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

```
int WholsFirst(int age1, int age2, int  
(*cmp)(int n1, int n2))  
{  
    return cmp(age1, age2);  
}
```

```
int OlderFirst(int age1, int age2)  
{  
    if(age1>age2)  
        return age1;  
    else if(age1<age2)  
        return age2;  
    else  
        return 0;  
}
```

```
int YoungerFirst(int age1, int age2)  
{  
    if(age1<age2)  
        return age1;  
    else if(age1>age2)  
        return age2;  
    else  
        return 0;  
}
```

```
int main(void)
```

```
{  
    int age1=20;  
    int age2=30;  
    int first;  
  
    printf("입장순서 1 \n");  
    first=WholsFirst(age1, age2,  
OlderFirst);  
    printf("%d세와 %d세 중 %d세가 먼  
저 입장! \n\n", age1, age2, first);  
  
    printf("입장순서 2 \n");  
    first=WholsFirst(age1, age2,  
YoungerFirst);  
    printf("%d세와 %d세 중 %d세가 먼  
저 입장! \n\n", age1, age2, first);  
    return 0;  
}
```

## 실습 2-2

**\* 함수 포인터를 쓰지 않는다면??**

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

```
int WholsFirst(int age1, int age2, int  
(*cmp)(int n1, int n2))  
{  
    return cmp(age1, age2);  
}
```

```
int OlderFirst(int age1, int age2)  
{  
    if(age1>age2)  
        return age1;  
    else if(age1<age2)  
        return age2;  
    else  
        return 0;  
}
```

```
int YoungerFirst(int age1, int age2)  
{  
    if(age1<age2)  
        return age1;  
    else if(age1>age2)  
        return age2;  
    else  
        return 0;  
}
```

```
int main(void)
```

```
{  
    int age1=20;  
    int age2=30;  
    int first;  
  
    printf("입장순서 1 \n");  
    first=WholsFirst(age1, age2,  
OlderFirst);  
    printf("%d세와 %d세 중 %d세가 먼  
저 입장! \n\n", age1, age2, first);  
  
    printf("입장순서 2 \n");  
    first=WholsFirst(age1, age2,  
YoungerFirst);  
    printf("%d세와 %d세 중 %d세가 먼  
저 입장! \n\n", age1, age2, first);  
    return 0;  
}
```



```
void SoSimpleFunc(void)
{
    printf("I'm so simple");
}

int main(void)
{
    int num=20;
    void * ptr;

    ptr=&num;    // 변수 num의 주소 값 저장
    printf("%p \n", ptr);

    ptr=SoSimpleFunc;    // 함수 SoSimpleFunc의 주소 값 저장
    printf("%p \n", ptr);
    return 0;
}
```

# 형(Type)이 존재하지 않는 void 포인터

```
void * ptr;
```

어떠한 주소 값도 저장이 가능한 void형 포인터

형 정보가 존재하지 않는 포인터 변수이기에 어떠한 주소 값도 저장이 가능하다.  
형 정보가 존재하지 않기 때문에 메모리 접근을 위한 \* 연산은 불가능하다.

```
void SoSimpleFunc(void)
{
    printf("I'm so simple");
}

int main(void)
{
    int num=20;
    void * ptr;

    ptr=&num;    // 변수 num의 주소 값 저장
    printf("%p \n", ptr);

    ptr=SoSimpleFunc;    // 함수 SoSimpleFunc의 주소 값 저장
    printf("%p \n", ptr);
    return 0;
}
```

```
int main(void)
{
    int num=20;
    void * ptr=&num;
    *ptr=20;    // 컴파일 에러!
    ptr++;    // 컴파일 에러!
    . . . .
}
```

형 정보가 존재하지 않으므로!!

001AF974

00F61109

실행결과



## Chapter 19-2. main 함수로의 인자 전달

```
int main(int argc, char *argv[])
{
    int i=0;
    printf("전달된 문자열의 수: %d \n", argc);

    for(i=0; i<argc; i++)
        printf("%d번째 문자열: %s \n", i+1, argv[i]);
    return 0;
}
```

\*.exe file 실행

---

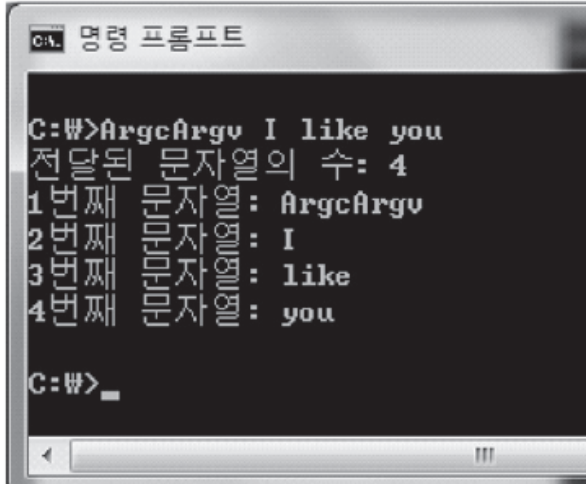


# main 함수를 통한 인자의 전달

---

```
int main(int argc, char *argv[])
{
    int i=0;
    printf("전달된 문자열의 수: %d \n", argc);

    for(i=0; i<argc; i++)
        printf("%d번째 문자열: %s \n", i+1, argv[i]);
    return 0;
}
```



```
C:\>ArgcArgv I like you
전달된 문자열의 수: 4
1번째 문자열: ArgcArgv
2번째 문자열: I
3번째 문자열: like
4번째 문자열: you
C:\>
```

인자를 전달하는 방식



# char \* argv[]

---

```
void SimpleFunc(TYPE * arr) { . . . . }  
void SimpleFunc(TYPE arr[]) { . . . . }
```

매개 변수 선언에서는 예외적으로 **\*arr**을 **arr[]**으로 대신할 수 있다!  
앞서 두 차례 확인한 내용!



그대로 적용한다.

```
void SimpleFunc(char **arr) { . . . . }  
void SimpleFunc(char * arr[]) { . . . . }
```

즉, char \* arr[]는 char형 이중 포인터이다.



# char \* argv[] 관련 예제

---

```
void ShowAllString(int argc, char * argv[])
{
    int i;
    for(i=0; i<argc; i++)
        printf("%s \n", argv[i]);
}

int main(void)
{
    char * str[3]={
        "C Programming",
        "C++ Programming",
        "JAVA Programming"
    };
    ShowAllString(3, str);
    return 0;
}
```

문자열의 주소 값을 모은 배열이므로 char형 포인터 배열을 선언!

str의 포인터 형은 char\*\*

```
C Programming
C++ Programming
JAVA Programming
```

실행결과



```
int main(int argc, char *argv[])
{
    int i=0;
    printf("전달된 문자열의 수: %d \n", argc);

    while(argv[i]!=NULL)
    {
        printf("%d번째 문자열: %s \n", i+1, argv[i]);
        i++;
    }
    return 0;
}
```





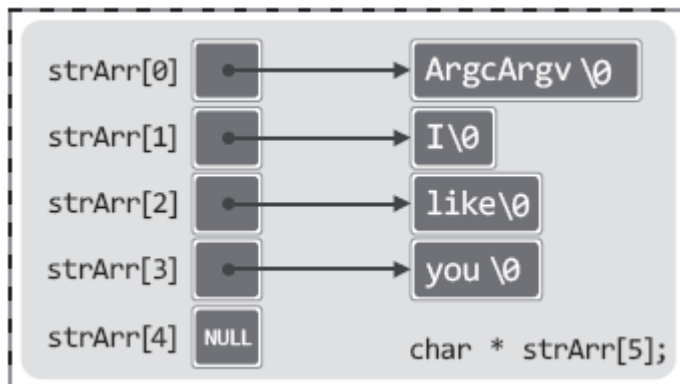
# 인자의 형성과정

c:\>ArgvArgv I like you

문자열의 구분

문자열 1	"ArgvArgv"
문자열 2	"I"
문자열 3	"like"
문자열 4	"you"

문자열의 구성



```
int main(int argc, char *argv[])
{
    int i=0;
    printf("전달된 문자열의 수: %d \n", argc);

    while(argv[i]!=NULL)
    {
        printf("%d번째 문자열: %s \n", i+1, argv[i]);
        i++;
    }
    return 0;
}
```

C:\> ArgvEndNULL "I love you"

전달된 문자열의 수: 2

1번째 문자열: ArgvEndNULL

2번째 문자열: I love you

실행결과

문자열 기반 함수의 호출

main(4, strArr);

# 실습

## 도전 3

프로그램을 구현하다 보면 난수(Random Number)를 발생시켜야 하는 경우가 종종 있다. 여기서 말하는 난수란 임의의, 정해지지 않은, 무엇이 될지 모르는 수를 의미한다. 그런데 다행인 것은 ANSI 표준에서 난수를 생성할 때 호출할 수 있는 다음 함수를 제공하고 있다는 것이다.

```
#include <stdlib.h>
int rand(void); // 의사 난수(pseudo-random number)를 반환
```

ANSI 표준에서는 이렇게 난수를 생성할 때 사용할 수 있는 함수 rand를 제공하고 있다. 이 함수의 사용방법은 다음과 같다.

### ❖ RandomNum.c

```
1. #include <stdio.h>
2. #include <stdlib.h>
3.
4. int main(void)
5. {
6.     int i;
7.     printf("난수의 범위: 0부터 %d까지 \n", RAND_MAX);
8.     for(i=0; i<5; i++)
9.         printf("난수 출력: %d \n", rand());
10.    return 0;
11. }
```

### 해설

- 7행: stdlib.h에 선언되어 있는 상수 RAND\_MAX를 출력하고 있다. 이 값은 생성될 수 있는 난수의 최대값을 의미한다. 즉, rand 함수는 0이상 RAND\_MAX 이하의 값을 반환한다.

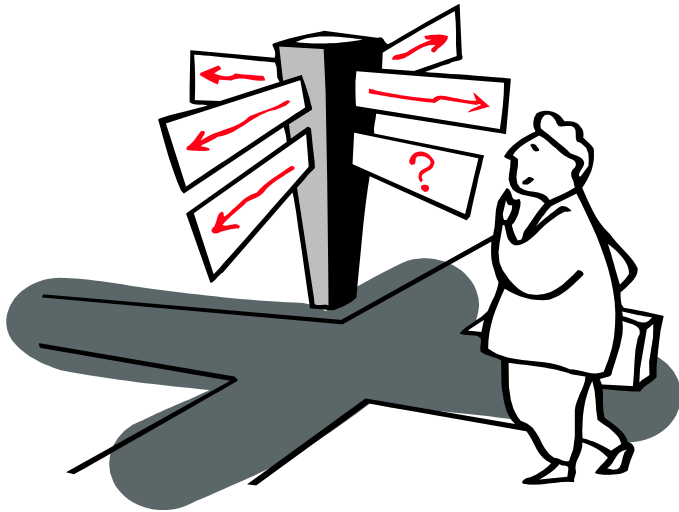
### ❖ 실행결과: RandomNum.c

command prompt

```
난수의 범위: 0부터 32767까지
난수 출력: 41
난수 출력: 18467
난수 출력: 6334
난수 출력: 26500
난수 출력: 19169
```

Calculate mean and variance

그럼 이어서 문제를 제시하겠다. 위의 예제에서는 0이상 RAND\_MAX 이하의 난수를 총 5개 생성하고 있다. 이 예제를 적절히 변경해서 0 이상 99 이하의 난수를 총 5개 생성하는 프로그램을 작성해보자(힌트: % 연산자를 적절히 활용하면 된다).



Chapter 19가 끝났습니다. 질문 있으신  
지요?