11주차 2차시 함수 포인터와 void 포인터

【학습목표】

1. void 포인터에 대해 설명할 수 있다.
2. main 함수로의 인자전달에 대해 설명할 수 있다.

학습내용1 : 함수 포인터와 void 포인터

1. 함수 포인터의 이해

가. 함수 포인터

- 함수의 이름은 함수가 저장된 메모리 공간을 가리키는 포인터이다(함수 포인터).
- 함수의 이름이 의미하는 주소 값은 함수 포인터 변수를 선언해서 저장할 수 있다.
- 함수 포인터 변수를 선언하라면 함수 포인터의 형(type)을 알아야 한다.

나. 함수 포인터형

- 함수 포인터의 형 정보에는 반환형과 매개변수 선언에 대한 정보를 담기로 약속
- 즉, 함수의 반환형과 매개변수 선언이 동일한 두 함수의 함수 포인터 형은 일치한다.

다. 함수 포인터 형 결정

```
int SimpleFunc(int num)                          반환형 int, 매개변수 int형 1개
double ComplexFunc(double num1, double num2)     반환형 double, 매개변수 double형 2개
```

2. 적절한 함수 포인터 변수의 선언

함수 포인터 변수를 선언하는 방법

```
int (*fptr) (int)      fptr은 포인터!
int (*fptr) (int)      반환형이 int인 함수 포인터!
int (*fptr) (int)      매개변수 선언이 int 하나인 함수 포인터!
```

```
int SoSimple(int num1, int num2) { . . . . }
int (*fptr) (int, int);   SoSimple함수이름과동일한형의 변수선언
fptr=SoSimple; 상수의 값을 변수에 저장
fptr(3, 4);    // SoSimple(3, 4)와 동일한 결과를 보임
```

함수 포인터 변수에 저장된 값을 통해서도 함수호출가능!

3. 함수포인터 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
```

4. 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;
}
```

```
001AF974
00F61109
```

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

형 정보가 존재하지 않으므로!!

학습내용2 : main 함수로의 인자전달

1. 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;
}
```

인자를 전달하는 방식

2. char *argv[]

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

매개 변수 선언에서는 예외적으로 *arr을 arr[]으로 대신할 수 있다!

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

즉, char * arr[]는 char형 이중 포인터이다.

3. 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
```

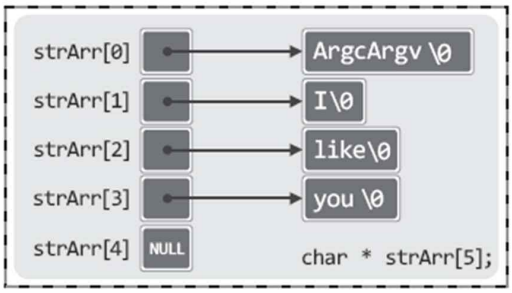
4. 인자의 형성과정

```
c:\>ArgcArgv I like you
```

문자열의 구분

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

문자열의 구성



문자열 기반 함수의 호출

```
main(4, strArr);
```

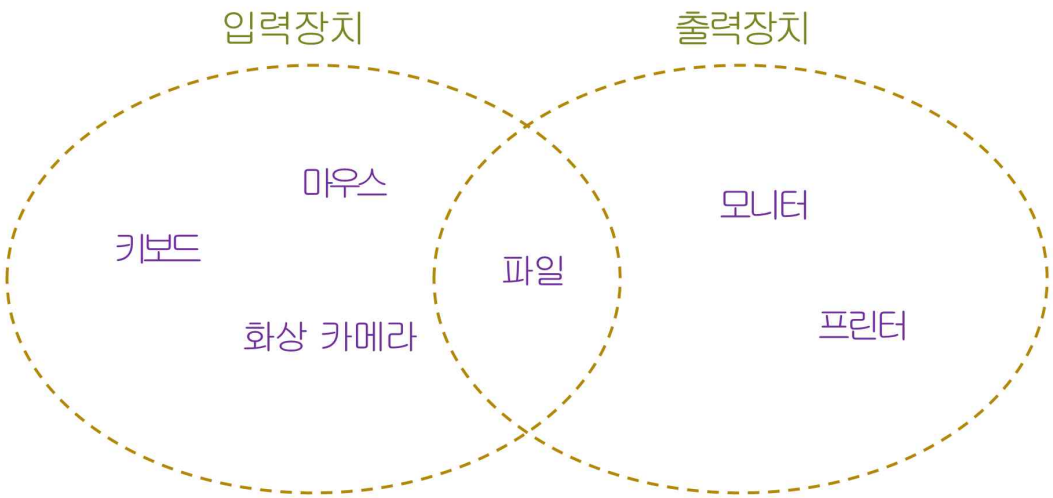
```
int main(int argc, char *argv[])
{
    int i=0;
    printf("전달된 문자열의 수: %d \n", argc);

    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
```

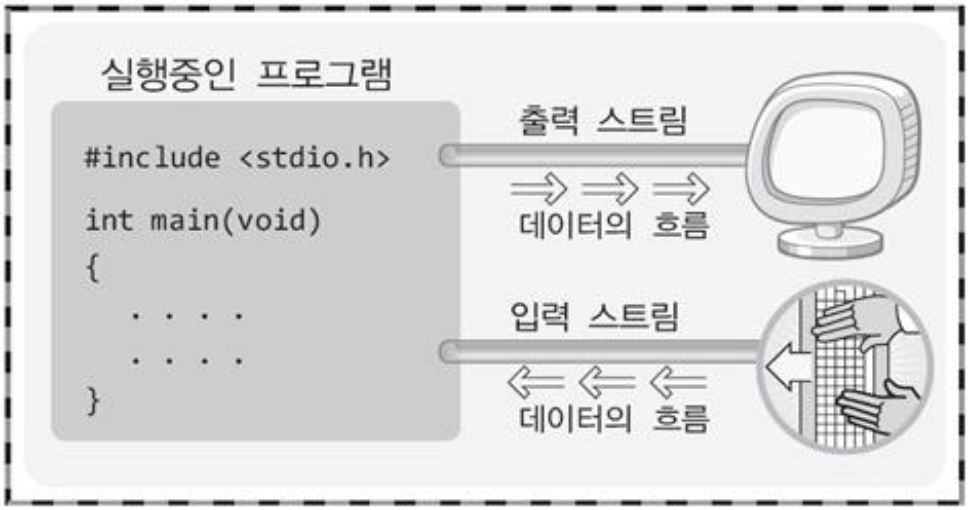
5. 스트림과 데이터의 이동

- 무엇이 입력이고 무엇이 출력인가?



입출력 장치는 매우 포괄적이다.
데이터를 컴퓨터 내부로 받아들이는 것이 입력이고 외부로 전송하는 것이 출력이다

- 데이터의 이동 수단이 되는 스트림



콘솔 입출력을 위한 스트림은 프로그램이 시작되면 OS에 의해서 자동으로 생성된다.

✈ 데이터의 입출력이 가능한 이유!
출력의 경로가 되는 출력 스트림과 입력의 경로가 되는 입력 스트림이 존재하기 때문

✈ 입출력 스트림이란?
OS가 데이터의 입출력을 위해 놓아주는 소프트웨어적인 형태의 다리!

- 스트림의 생성과 소멸

• stdin	표준 입력 스트림	키보드 대상으로 입력
• stdout	표준 출력 스트림	모니터 대상으로 출력
• stderr	표준 에러 스트림	모니터 대상으로 출력

✓ stdin과 stdout은 각각 표준 입력 스트림과 표준 출력 스트림을 의미하는 이름들이다.
✓ stderr은 표준 에러 스트림이라 하며, 출력의 대상은 stdout과 마찬가지로 모니터이다.
✓ 출력 리다이렉션이라는 것을 통해서 stdout과 stderr이 향하는 데이터 전송의 방향을 각각 달리 할 수 있다.
✓ stdin, stdout, stderr은 모두 프로그램 시작과 동시에 자동으로 형성되고 프로그램 종료시 자동으로 소멸된다.
✓ 이외의 스트림들은 프로그래머가 직접 형성해야 한다. 예를 들어 파일 입출력을 위한 스트림은 직접 형성해야 한다.
스트림이라 불리는 이유는 데이터의 이동을 한 방향으로만 형성하기 때문이다.
물이 한 방향으로 흐르듯 스트림도(스트림은 물의 흐름을 의미함)
한 방향으로만 데이터가 이동한다.

【학습정리】

1. 형(Type)이 존재하지 않는 void 포인터는 형 정보가 존재하지 않기 때문에 메모리 접근을 위한 * 연산은 불가능하다.

2. 함수 포인터형 정보에는 반환형과 매개변수 선언에 대한 정보를 담기로 약속되어 있다. 즉, 함수의 반환형과 매개변수 선언이 동일한 두 함수의 함수 포인터 형은 일치한다.

3. 스트림이라 불리는 이유는 데이터의 이동을 한 방향으로만 형성하기 때문이다. 물이 한 방향으로 흐르듯 스트림도(스트림은 물의 흐름을 의미함) 한 방향으로만 데이터가 이동한다.