1 포인터 개념

1 포인터 개념

포인터란?

사용하는 모든 변수는 메모리의 특정 위치에 저장되는데 그 위치를 나타내는 **메모리 주소**

포인터변수

- 메모리의 주소 값을 저장하는 특별한 변수
- 포인터 변수가 어떤 변수의 주소를 저장하고 있다는 것은 그 변수를 가리키고 있다는 의미
- 연결된 주소의 변수 영역 액세스 가능
- 포인터 변수를 간단히 포인터라고 함

2 포인터 엑세스 처리

- 편지봉투에 받는 사람의 집주소를 쓰면 그 주소로 편지가 전달되어 집주인이 편지를 받게 됨

※ 출처 : C로 배우는 쉬운 자료구조(개정3판), 이지영, 한빛미디어

2 포인터 엑세스 처리

- 편지봉투를 포인터라고 생각하면 편지봉투에 쓰는 **'받는 사람주소'**는 포인터에 저장된 변수의 **메모리 주소**가 되고, 주소에 해당하는 **집주인이 편지를 받은 것**은 포인터를 통한 **엑세리** 처리로 생각할 수 있음

※ 출처 : C로 배우는 쉬운 자료구조(개정3판), 이지영, 한빛미디어

3 포인터 사용 예시

- 포인터를 사용해 다른 변수 액세스 가능
  - int i에서 할당된 주소를 150이라고 하면 int *ptr=&i 에 의해 변수 i의 주소 150을 포인터 ptr에 저장하므로 포인터 ptr은 변수 i를 가리킴
  - 그러면 변수 i와 포인터 ptr이 논리적으로 서로 연결되어 ptr을 사용하여 변수 i를 액세스 할 수 있음

```
int i;
int *ptr = &i;
```

※ 출처 : C로 배우는 쉬운 자료구조(개정3판), 이지영, 한빛미디어

4 포인터 선언

포인터 선언 형식

자료형 ①

*포인터이름; ②

- 포인터 자체의 자료형이 아니라 포인터에 저장할 수 있는 일반 변수의 자료형
- 일반 변수 이름과 구별되도록 앞에 *를 붙여 포인터임을 나타냄

4 포인터 선언

- 포인터를 다양한 자료형으로 선언한 예

  : 포인터는 자료형에 상관없이 포인터 자체의 크기는 **메모리** 주소 한 개의 크기인 **2바이트**

4 포인터 선언

▶ 포인터에서 선언한 자료형에 따른 메모리 액세스 범위 ※ 출처 : C로 배우는 쉬운 자료구조(개정3판), 이지영, 한빛미디어

```
char *ptr;
```

❶ 1바이트의 char형 변수의 주소를 저장할 포인터를 선언한 예이다.
ptr에 저장된 메모리 주소로부터 1바이트의 char 데이터를 액세스한다.

```
short *ptr;
```

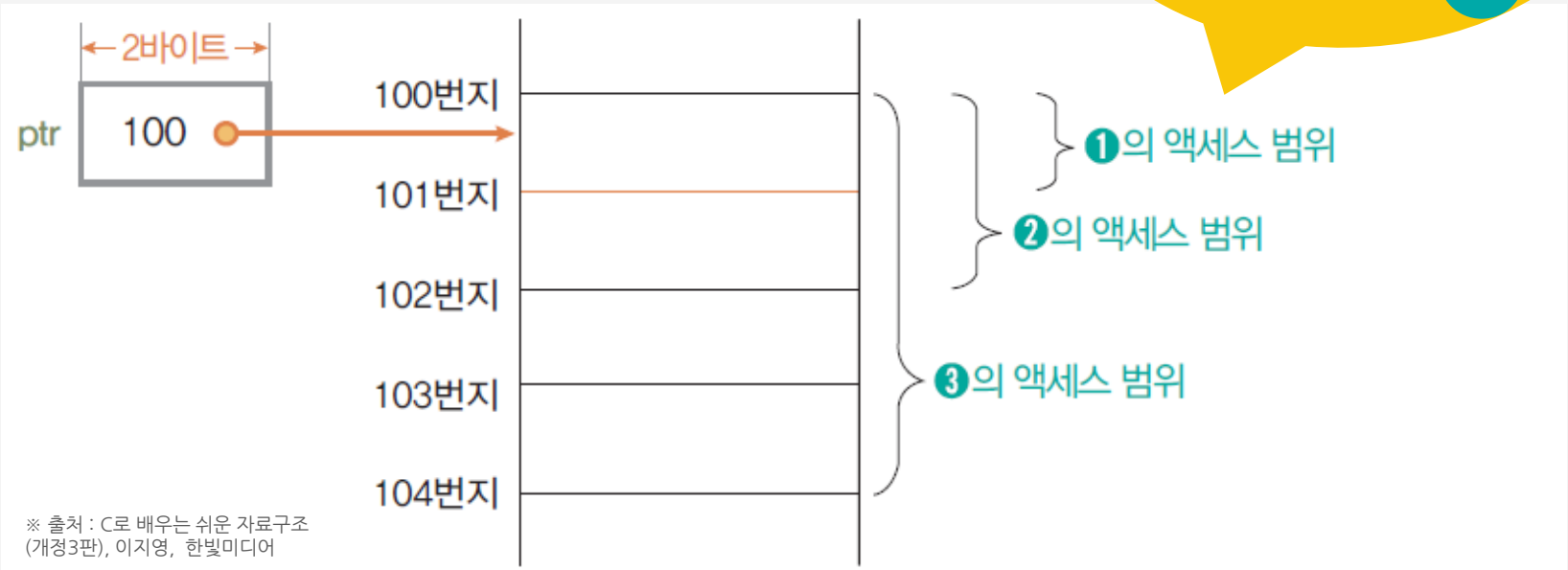
❷ 2바이트의 short형 변수의 주소를 저장할 포인터를 선언한 예이다.
ptr에 저장된 메모리 주소로부터 2바이트의 short 데이터를 액세스한다.

```
int *ptr;
```

❸ 4바이트의 int형 변수의 주소를 저장할 포인터를 선언한 예이다.
ptr에 저장된 메모리 주소로부터 4바이트의 int 데이터를 액세스한다.

4 포인터 선언

- 포인터에서 선언한 자료형에 따른 메모리 액세스 범위



※ 출처 : C로 배우는 쉬운 자료구조 (개정3판), 이지영, 한빛미디어

5 포인터 연산

- C언어에서 사용하는 포인터 관련 연산자에는 주소연산자 & 와 참조연산자 *가 있음

주소연산자	참조연산자
&	*

5 포인터 연산

주소연산자 &

- 변수의 주소를 얻기 위해 사용
- 변수 앞에 주소 연산자를 사용하면 그 변수의 주소를 사용 가능
  (주소 연산자를 사용할 변수와 포인터는 같은 자료형으로 선언되어 있어야 함)

```
포인터 = &변수;
```

5 포인터 연산

주소 연산자 사용 예

(a)

```
int i = 10;
int *ptr;
```

(b)

(c)

```
ptr = &i;
```

(d)

※ 출처 : C로 배우는 쉬운 자료구조(개정3판), 이지영, 한빛미디어

5 포인터 연산

- 참조 연산자 *
  - 다른 변수의 주소가 저장된 포인터에 참조 연산자를 사용하면 저장된 주소 영역(참조영역) 또는 주소에 있는 값(변수에 저장된 값)을 액세스할 수 있음

5 포인터 연산

참조연산자 사용 형식

- ①과 같이 지정연산자의 좌변에 있는 포인터에 참조 연산자를 사용하면 포인터가 가리키고 있는 주소 영역을 액세스하여 값을 저장할 수 있음
- ②와 같이 지정 연산자의 우변에 있는 포인터에 참조 연산자를 사용하면, 포인터가 가리키는 주소 영역에 잇는 값을 액세스하여 변수에 저장 할 수 있음

```
① *포인터 = 값;
② 변수 = *포인터;
```

5 포인터 연산

포인터 연산자를 사용한 예

```
int i, j;
int *ptr;
ptr = &i;
*ptr = 10;
j = *ptr;
```

❶ 주소 연산자를 사용하여 변수 i의 주소를 포인터 ptr에 할당한다. 포인터 ptr은 변수 i를 가리킨다.

❷ 참조 연산자를 사용하여 포인터 ptr이 가리키는 영역에 값 10을 지정한다. 따라서 변수 i에는 10이 저장된다.

❸ 다시 참조 연산자를 사용하여 ptr이 가리키는 영역의 값을 변수 j에 지정한다. 따라서 ptr이 가리키는 변수 i의 값인 10을 변수 j에 저장한다.

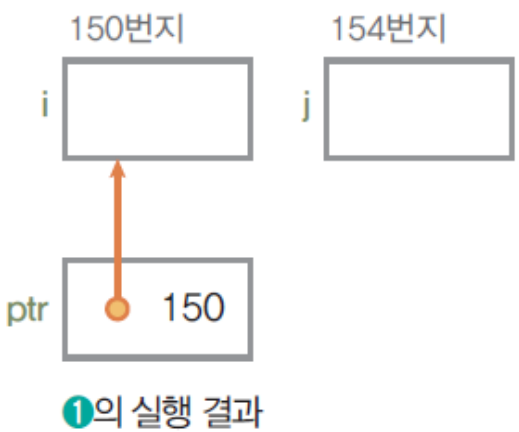
※ 출처 : C로 배우는 쉬운 자료구조 (개정3판), 이지영, 한빛미디어

5 포인터 연산

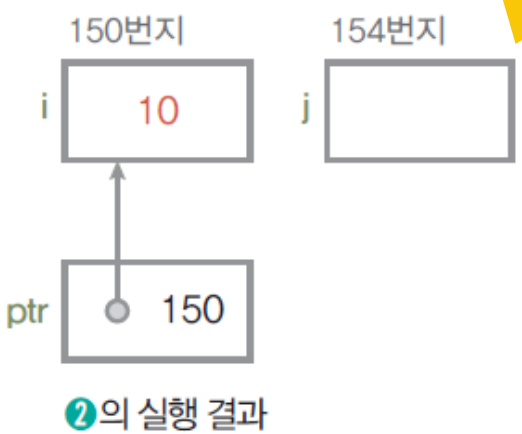
- 포인터 연산자를 사용한 예

```
int I, j;
int *ptr;
ptr =&i;   ------- 1
*ptr =10;  ------ 2
j =*ptr;   -------- 3
```

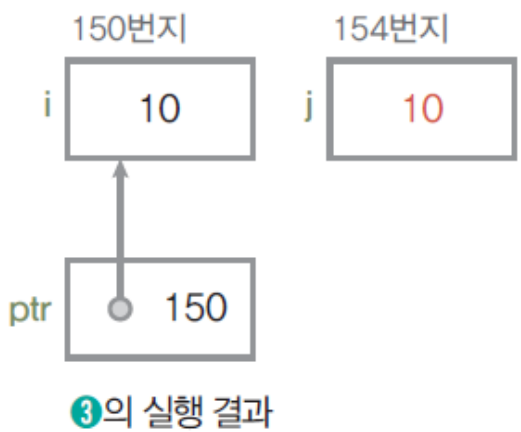
※ 출처 : C로 배우는 쉬운 자료구조 (개정3판), 이지영, 한빛미디어



❶의 실행 결과



❷의 실행 결과



❸의 실행 결과

5 포인터 연산

예제

포인터 연산자를 이용해
변수 액세스하기 소스

```
void main() {
    int i = 10, j = 20;
    int *ptr;
    printf("\n i의 값 = %d \n j의 값 = %d", i, j);
    printf("\n i의 메모리 주소(&i) = %u", &i);
    printf("\n j의 메모리 주소(&j) = %u", &j);
    ptr = &i;
    printf("\n\n << ptr=&i 실행 >>");
    printf("\n ptr의 메모리 주소(&ptr) = %u", &ptr);
    printf("\n ptr의 값(ptr) = %u", ptr);
    printf("\n ptr의 참조 값(*ptr) = %d", *ptr);
    ptr = &j;
    printf("\n\n << ptr=&j 실행 >>");
    printf("\n ptr의 메모리 주소(&ptr) = %u", &ptr);
    printf("\n ptr의 값(ptr) = %u", ptr);
    printf("\n ptr의 참조값(*ptr) = %d", *ptr);
    i = *ptr;
    printf("\n\n << i=*ptr 실행 >>");
    printf("\n i의 값 = %d", i);getchar();
}
```

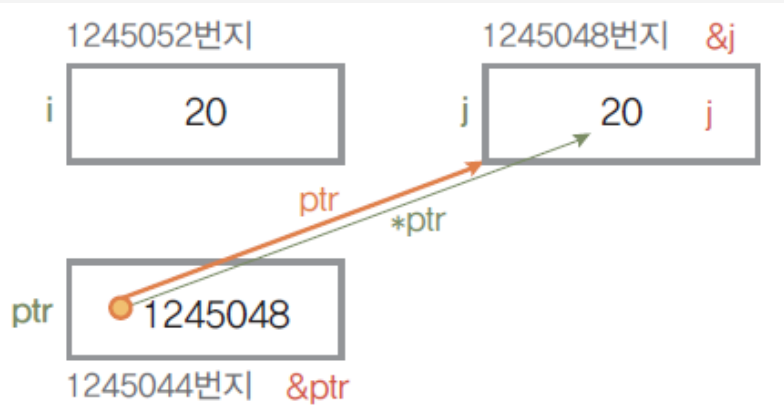
※ 출처 : C로 배우는 쉬운 자료구조(개정3판), 이지영, 한빛미디어

5 포인터 연산

예제

※ 출처 : C로 배우는 쉬운 자료구조 (개정3판), 이지영, 한빛미디어

포인터 연산자를 이용해
변수 액세스한 프로그램 실행 결과



명령 프롬프트

```
i의 값 = 10
j의 값 = 20
i의 메모리 주소(&i) = 1245052
j의 메모리 주소(&j) = 1245048

<< ptr=&i 실행 >>
ptr의 메모리 주소(&ptr) = 1245044
ptr의 값(ptr) = 1245052
ptr의 참조값(*ptr) = 10

<< ptr=&j 실행 >>
ptr의 메모리 주소(&ptr) = 1245044
ptr의 값(ptr) = 1245048
ptr의 참조값(*ptr) = 20

<< i=*ptr 실행 >>
i의 값 = 20
```

6 포인터의 초기화

※ 단, 초기값이 일반 데이터 값이 아니라 메모리 주소라는 사실을 주의해야 함

- 일반 변수를 초기화하는 방법

포인터에 주소 지정 방법

① 주소 연산자를 사용하여 변수 주소를 지정

```
int i;
int *ptr = &i;
```

※ 출처 : C로 배우는 쉬운 자료구조(개정3판), 이지영, 한빛미디어

6 포인터의 초기화

- 일반 변수를 초기화하는 방법

포인터에 주소 지정 방법

② 동적 메모리를 할당하고 그 시작 주소를 포인터값으로 지정

```
char *ptr = (char *)malloc(100);
```

동적메모리 할당은 malloc()함수를 사용하여 문자형(char)공간 100개를 할당하고 그 시작주소를 포인터 ptr에 지정함

※ 출처 : C로 배우는 쉬운 자료구조(개정3판), 이지영, 한빛미디어

6 포인터의 초기화

- 일반 변수를 초기화하는 방법

포인터에 주소 지정 방법

③ 문자형 포인터에 문자열의 시작 주소를 지정

```
char *ptr = "korea";
```

- 문자열 자료형은 일반 자료형과 다르게 문자열 지정만으로 그 시작 주소를 전달 할 수 있음
- “korea” 문자열이 포인터 ptr에 저장되는 것이 아니라 “korea”문자열이 저장된 위치의 시작주소가 포인터 ptr에 저장됨

※ 출처 : C로 배우는 쉬운 자료구조(개정3판), 이지영, 한빛미디어

6 포인터의 초기화

- 일반 변수를 초기화하는 방법

포인터에 주소 지정 방법

배열이름은 문자열과 마찬가지로
그 시작 주소를 전달할 수 있음

④ 배열 이름을 이용하여 배열 시작 주소를 지정

```
char A[100];
char *ptr = A;
```

※ 출처 : C로 배우는 쉬운 자료구조(개정3판), 이지영, 한빛미디어

6 포인터의 초기화

- 일반 변수를 초기화하는 방법

포인터에 주소 지정 방법

⑤ 배열의 첫 번째 요소(0번 인덱스)의 주소를 이용하여 배열 시작 주소를 지정

```
char A[100];
char *ptr = &A[0];
```

※ 출처 : C로 배우는 쉬운 자료구조 (개정3판), 이지영, 한빛미디어

2 문자배열과 포인터 배열 차이점

2 문자배열과 포인터 배열 차이점

1 포인터와 문자 배열

- 포인터를 이용하면 배열에 저장된 문자열에 대한 연산을 쉽게 처리할 수 있음

2 | 문자배열과 포인터 배열 차이점

1 포인터와 문자 배열

예제

포인터를 이용해
문자열 처리하기

```
void main() {
  int i;
  char string1[20] = "Dreams come true!", string2[20], *ptr1, *ptr2;
  ptr1 = string1;
  printf("\n string1의 주소 = %u \t ptr1 = %u", string1, ptr1);
  printf("\n string1 = %s \n ptr1 = %s", string1, ptr1);
  printf("\n\n %s", ptr1 + 7);
  ptr2 = &string1[7];      printf("\n %s \n\n ", ptr2);
  for (i = 16; i >= 0; i--)  putchar(*(ptr1 + i));
  for (i = 0; i<20; i++)   string2[i] = *(ptr1 + i);
  printf("\n\n string1 = %s", string1);
  printf("\n string2 = %s", string2);
  *ptr1 = 'P';              *(ptr1 + 1) = 'e';
  *(ptr1 + 2) = 'a';      *(ptr1 + 3) = 'c';
  *(ptr1 + 4) = 'e';       rintf("\n\n string1 = %s\n", string1);
  getchar();
}
```

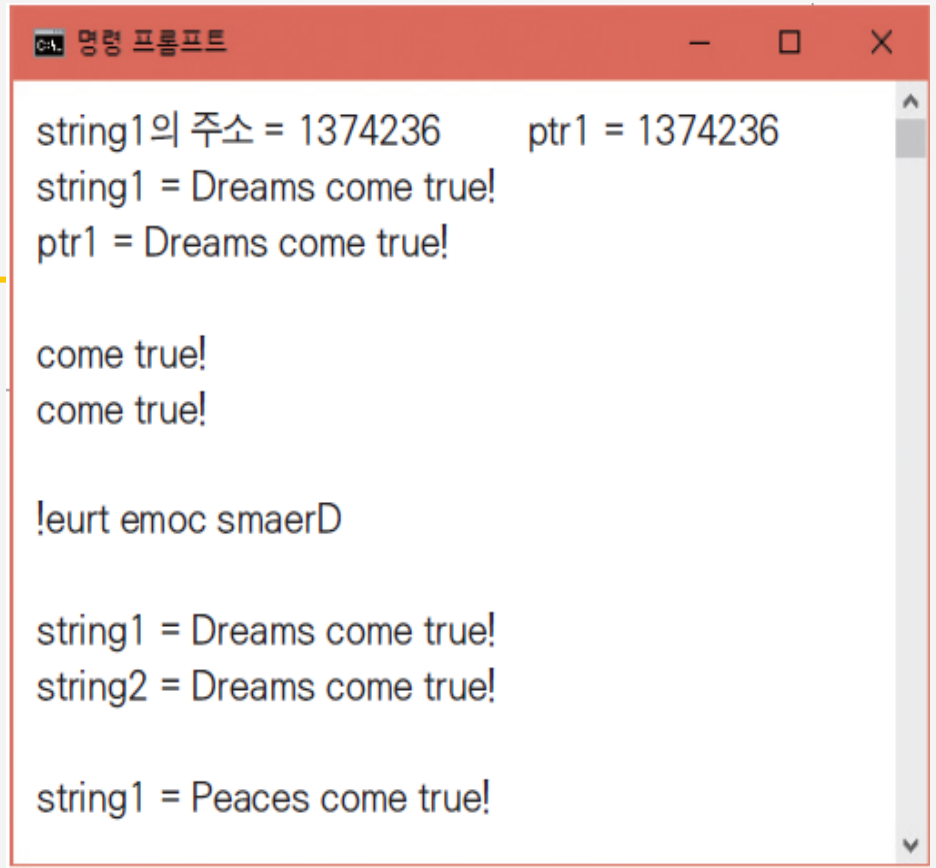
※ 출처 : C로 배우는 쉬운 자료구조(개정3판), 이지영, 한빛미디어

2 | 문자배열과 포인터 배열 차이점

1 포인터와 문자 배열

예제

포인터를 이용해
문자열 처리하기



※ 출처 : C로 배우는 쉬운 자료구조(개정3판), 이지영, 한빛미디어

2 포인터 배열

- 포인터 자료형을 배열로 구성
- 여러 개의 포인터를 하나의 배열로 구성한 배열의 특징과 포인터의 특징을 모두 활용할 수 있음
- 포인터 배열의 선언형식

```
자료형 *포인터배열이름 [배열크기];
```

2 포인터 배열

- 포인터 배열에서 각 배열요소는 포인터가 됨
- 2차원 문자배열을 1차원 포인터배열로 표현할 수 있음

2차원 배열의 행의 개수	포인터배열의 각 배열요소
포인터배열의 크기	각 문자열에 대한 시작주소를 가진 포인터

3 2차원 배열과 1차원 포인터배열의 비교

- (a)는 배열 선언 될 때 지정한 크기의 배열공간이 메모리에 할당됨
  - 한번 할당된 배열의 크기를 줄이거나 늘리기 어렵기 때문에 처음 선언한 배열크기보다 실제 사용 공간이 작으면 메모리가 낭비되고, 실제 필요한 공간이 할당 크기보다 크면 배열을 새로 만들어야 하는 문제가 발생할 수 있음

(a) `char string[3][10] = { "Dreams", "come", "true!" };`

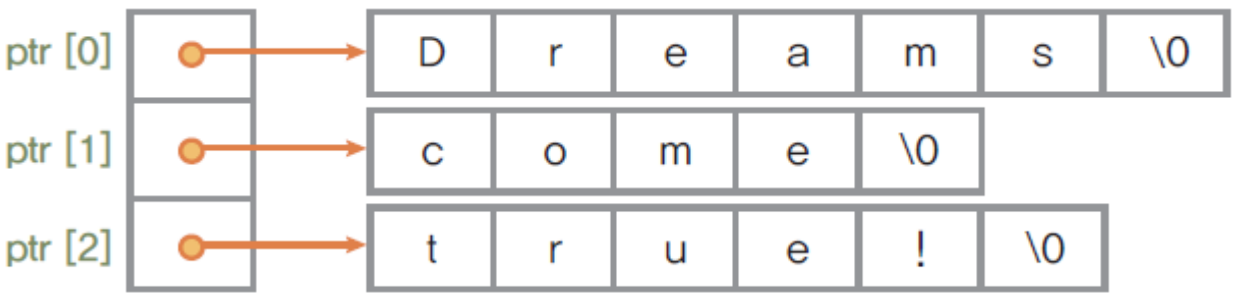
	[0]	[1]	[2]	[3]	[4]	[5]	[6]	[7]	[8]	[9]
string[0]	D	r	e	a	m	s	\0			
string[1]	c	o	m	e	\0					
string[2]	t	r	u	e	!	\0				

※ 출처 : C로 배우는 쉬운 자료구조(개정3판), 이지영, 한빛미디어

3 2차원 배열과 1차원 포인터배열의 비교

- (b) 포인터 배열을 사용하여 문자열을 저장하면 저장하는 문자열의 길이에 따라 메모리가 할당됨
  - 그러므로 저장할 문자열의 길이를 정확히 예측할 수 없거나 문자열의 길이가 자주 변하는 경우에 메모리를 좀 더 효율적으로 사용할 수 있음

```
(b) char *ptr[3] = { { "Dreams" },{ "come" },{ "true!" } };
```



※ 출처 : C로 배우는 쉬운 자료구조(개정3판), 이지영, 한빛미디어

3 2차원 배열과 1차원 포인터배열의 비교

예제

포인터 배열을 이용해 문자열 저장하기

```
void main() {
    int i;
    char *ptrArray[4] = { { "Korea" },{ "Seoul" },{ "Mapo" },{ "152번지 2
/ 3" } };
    for (i = 0; i<4; i++)  printf("\n %s", ptrArray[i]);
    ptrArray[2] = "Jongno";
    printf("\n\n");
   for (i = 0; i<4; i++)    printf("\n %s", ptrArray[i]);
   getchar();
}
```

※ 출처 : C로 배우는 쉬운 자료구조(개정3판), 이지영, 한빛미디어

3 2차원 배열과 1차원 포인터배열의 비교

예제

포인터 배열을 이용해 문자열 저장하기

※ 출처 : C로 배우는 쉬운 자료구조(개정3판), 이지영, 한빛미디어

3 포인터의 포인터(이중 포인터) 개념

3 | 포인터의 포인터(이중 포인터) 개념

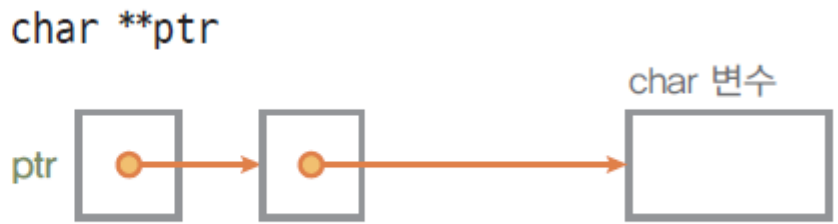
1 포인터의 포인터

- 포인터를 가리키고 있는 포인터 : 이중 포인터(**)
- 일반변수의 주소가 아니라 포인터의 주소를 가지고 있는 포인터를 의미함
- 포인터의 포인터 선언 형식과 예

※ 출처 : C로 배우는 쉬운 자료구조(개정3판), 이지영, 한빛미디어

```
자료형 **포인터이름;
```

(a) 선언 형식



(b) 사용 예

3 | 포인터의 포인터(이중 포인터) 개념

4 포인터의 포인터

예제

포인터 배열과 포인터의 포인터 사용하기

```
void main() {
char *ptrArray[2];          char **ptrptr;        int i;
ptrArray[0] = "Korea"; ptrArray[1] = "Seoul";    ptrptr = ptrArray;
printf("\n ptrArray[0]의 주소 (&ptrArray[0]) = %u", &ptrArray[0]);
printf("\n ptrArray[0]의 값 (ptrArray[0]) = %u", ptrArray[0]);
printf("\n ptrArray[0]의 참조값 (*ptrArray[0]) = %c", *ptrArray[0]);
printf("\n ptrArray[0]의 참조문자열 (*ptrArray[0]) = %s \n",
*ptrArray);
printf("\n ptrArray[1]의 주소 (&ptrArray[1]) = %u", &ptrArray[1]);
printf("\n ptrArray[1]의 값 (ptrArray[1]) = %u", ptrArray[1]);
printf("\n ptrArray[1]의 참조값 (*ptrArray[1]) = %c", *ptrArray[1]);
printf("\n ptrArray[1]의 참조문자열(*ptrArray[1])= %s \n",
*(ptrArray + 1));
printf("\n ptrptr의 주소 (&ptrptr) = %u", &ptrptr);
printf("\n ptrptr의 값 (ptrptr) = %u", ptrptr);
printf("\n ptrptr의 1차 참조값 ( *ptrptr) = %u", *ptrptr);
printf("\n ptrptr의 2차 참조값 (**ptrptr) = %c", **ptrptr);
printf("\n ptrptr의 2차 참조문자열 (**ptrptr) = %s", *ptrptr);
printf("\n\n *ptrArray[0] : ");
for (i = 0; i<5; i++)  printf("%c", *(ptrArray[0] + i));
printf("\n **ptrptr : ");
for (i = 0; i<5; i++)   printf("%c", *(*ptrptr + i));
printf("\n\n *ptrArray[1] : ");
for (i = 0; i<5; i++)   printf("%c", *(ptrArray[1] + i));
printf("\n **(ptrptr+1) : ");
for (i = 0; i<5; i++)  printf("%c", *(*(ptrptr + 1) + i));
getchar();
}
```

※ 출처 : C로 배우는 쉬운 자료구조 (개정3판), 이지영, 한빛미디어

3 | 포인터의 포인터(이중 포인터) 개념

4 포인터의 포인터

예제

포인터 배열과 포인터의 포인터 사용하기

```
명령 프롬프트
ptrArray[0]의 주소 (&ptrArray[0]) = 1245048
ptrArray[0]의 값 (ptrArray[0]) = 4338436
ptrArray[0]의 참조값 (*ptrArray[0]) = K
ptrArray[0]의 참조 문자열 (*ptrArray[0]) = Korea

ptrArray[1]의 주소 (&ptrArray[1]) = 1245052
ptrArray[1]의 값 (ptrArray[1]) = 4338428
ptrArray[1]의 참조값 (*ptrArray[1]) = S
ptrArray[1]의 참조 문자열 (*ptrArray[1])= Seoul

ptrptr의 주소 (&ptrptr) = 1245044
ptrptr의 값 (ptrptr) = 1245048
ptrptr의 1차 참조값 ( *ptrptr) = 4338436
ptrptr의 2차 참조값 (**ptrptr) = K
ptrptr의 2차 참조 문자열 (**ptrptr) = Kore

*ptrArray[0] : Korea
**ptrptr : Korea

*ptrArray[1] : Seoul
**(ptrptr+1) : Seoul
```

※ 출처 : C로 배우는 쉬운 자료구조(개정3판), 이지영, 한빛미디어