@ 포인터 변수가 참조하는 대상을 변경하지 못하게 하는 const

Int mian(void){

Int num = 20;

Const int \*ptr = &num // ptr에 저장된 값 변경을 막겠다.

//const 는 선언하는 위치에 따라 약간의 차이가 있음

\*ptr = 30; // 컴파일 에러

Num = 40; // 컴파일 성공

Int main (void) {

Int num1 = 20;

Int num2 = 30;

Int \*const ptr = &num1;

Ptr = &num2; // 컴파일 에러

\*ptr = 40; // 컴파일 성공

Const int \* const ptr = &num;

\*ptr = 20; // 컴파일 에러

Ptr = &age; // 컴파일 에러

@ const 선언이 갖는 의미

Const 선언은 많이 하면 그만큼 프로그램 코드의 안전성이 높아진다.

Int main(void) {

Double PI = 3.1415;

Double rad;

PI = 3.07; // 실수로 잘못 삽입된 문장, 컴파일 시 발견이 안됨

Int main(void){

Const double PI = 3.1415;

Double rad;

PI = 3.07; // 컴파일 시 오류가 발견됨 \*\* 코드의 안전성 증가 !! \*\*

@ 포인터의 포인터

포인터의 포인터는 포인터 변수를 가르키는 또 다른 포인터 변수를 뜻한다. 흔히 “이중 포인터” 또는 “더블 포인터” 라고 한다.

Int \*\*dptr; // int 형 이중 포인터

Int main(void){

Double num = 3.14;

Double \*ptr = &num;

Double \*\*dptr = &ptr;

. . . . .

}

\*dptr = . . . . ; // \*dtpr은 포인터 변수 ptr을 의미함

\*(\*dptr) = . . . . . ; // \*(\*dptr)은 변수 num을 의미함

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| dptr | -----------------> | ptr | -----------------> | Num = 3.14 |

#include <stdio.h>

Int main(void){

Double num = 3.14;

Double \*ptr = &num;

Double \*\*dptr = &ptr;

Double \*ptr2;

Printf(“%u %u \n”, ptr, \*dptr);

Printf(“%u %u \n”, num, \*\*dptr); //동일한 출력 결과가 나온다.

Ptr2= \*dptr; //ptr2 = ptr 과 동일한 문장.

\*ptr2 = 10.99;

Printf(“%u %u \n”, num, \*\*dptr);

Return 0;

}

실행 결과 0032FD00 0032FD00

3.14 3.14

10.99 10.99

@ 포인터 변수대상의 call – by reference

두 변수가 아닌(두 싱글 포인터 변수)에 저장된 값을 서로 바꾸어 저장하는 함수를 정의해본다.

!! 잘못된 예

#include (stdio.h>

Void swapIntPtr(int \*p1, int \*p2){

Int \*temp = p1;

P1 = p2;

P2 = temp;

}

Int main(void){

Int num1 = 10, num2 = 20;

Int \*ptr1, \*ptr2;

Ptr1 = &num1, ptr2 = &num2

Printf(“\*ptr1, \*ptr2 : %d %d \n”, \*ptr1, \*ptr2);

SwapintPtr(ptr1, ptr2);

Printf(“\*ptr1, \*ptr2 : %d %d \n”, \*ptr1, \*ptr2);

Return 0;

}

실행 결과 \*ptr1, \*ptr2 : 10, 20

\*ptr1, \*ptr2 : 10, 20

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 0x10  Ptr1 | ------------->  대입 | 0x20  P1 | ------------->  참조 | 10  Num1 |
|  |  |  |  |  |
| 0x20  Ptr2 | ------------->  대입 | 0x10  P2 | ------------->  참조 | 20  Num2 |

P1과 P2에 저장된 값은 바뀌지만 ptr1과 ptr2는 별개의 변수이기 때문이다.

!! 바른 예

#include <stdio.h>

Void swapIntPtr(int \*\*dp1, int \*\*dp2){

Int \*temp = \*dp1;

\*dp1 = \*dp2;

\*dp2 = temp;

}

Int main(void){

Int num1 = 10, num2 = 20;

Int \*ptr1, \*ptr2;

Ptr1 = &num1, ptr2 = &num2;

Printf(“\*ptr1, \*ptr2 : %d %d \n”, \*ptr1, \*ptr2);

SwapIntPtr(“&ptr1, &ptr2); //ptr1과 ptr2의 주소값 전달

Printf(“\*ptr1, \*ptr2 : %d %d \n”, \*ptr1, \*ptr2);

Return 0;

}

실행 결과 \*ptr1, \*ptr2 : 10, 20

\*ptr1, \*ptr2 : 20, 10

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Dp1 | -------------> | Ptr1 | -------------> | Num1 = 10 |
|  |  |  |  |  |
| Dp2 | -------------> | Ptr2 | -------------> | Num2 =20 |

Ptr1과 ptr2의 주소값이 SwapIntPtr의 매개변수로 전달

@ 포인터 배열과 포인터 배열의 포인터형

#include<stdio.h>

Int main(void){

Int num1 = 10, num2 = 20, num3 = 30;

Int \*ptr1 = &num1;

Int \*ptr2 = &num2;

Int \*ptr3 = &num3;

Int \*ptrArr[] = { ptr1, ptr2, ptr3};

Int \*\*dptr = ptrArr[]; // 대입 연산이 허용된다는 것은 포인터형이 일치하기 때문.

Printf(“%d %d %d \n”, \*(ptrArr[0]), \*(ptrArr[1]), \*(ptrArr[2]));

Printf(“%d %d %d \n”, \*(dptr[0]), \*(dptr[1]), \*(dptr[2]));

Return 0;

}

실행 결과 10 20 30

10 20 30

@ 다차원 배열과 포인터의 관계

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Int arr2d[3][3] | | | | | | | | |
| arr2d -----------> | 1열 | 2열 | 3열 | 1행  2행  3행 | arr2d[0]---------->  arr2d[1]---------->  arr2d[2]----------> | 1열 | 2열 | 3열 |
|  |  |  |  |  |  |
|  |  |  |  |  |  |
|  |  |  |  |  |  |

Int arr2d[3][3];

Printf(“%d %d %d \n”, arr2d, arr2d[0], &arr2d[0][0]);

Printf(“%d %d \n”, arr2d[1], &arr2d[1][0]);

Printf(“%d %d \n”, arr2d[2], &arr2d[2][0]);

Printf(“sizeof(arr2d): %d \n”, sizeof(arr2d));

Printf(“sizeof(arr2d[0] : %d \n”, sizeof(arr2d[0]));

Printf(“sizeof(arr2d[1] : %d \n”, sizeof(arr2d[1]));

Return 0;

}

실행결과

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 4585464 | 4585464 | 4585464 |  | 실행 때마다 주소는 바뀜  메모리에 랜덤 엑세스  실행 때마다 다른 위치에 할당 |
| 4585476 | 4585476 |  |  |
| 4585488 | 4585488 |  |  |
| Sizeof(arr2d) : 36 | | |  |  |
| Sizeof(arr2d[0]) : 12 | | |  | 크기가 다름 |
| Sizeof(arr2d[1]) : 12 | | |  |  |

arr2d는 첫번째 요소를 가르키면서 배열 전체를 의미함,

arr2d[0]은 첫번째 요소를 가르키는 것은 같되, 1행만을 의미함

@ 배열이름 기반의 포인터 연산

Int iarr[3];

Double darr[8]

Printf(“%p”, iarr+1); //int형 포인터 iarr+sizeof(int)가 나옴

Printf(“%p”, darr+1); //double형 포인터 darr+sizeof(double)이 나옴

“두 포인터형이 같다면, 포인터를 대상으로 하는 증가, 감소하는 값의 크기는 동일하다.

#include <stdio.h>

Int mian(void){

Int arr1[3][2];

Int arr2[2][3];

Printf(“arr1 : %p \n”, arr1);

Printf(“arr1+1 : %p \n”, arr1+1);

Printf(“arr1+2 : %p \n”, arr1+2);

Printf(“arr2 : %p \n”, arr2);

Printf(“arr2+1 : %p \n”, arr2+1);

Return 0;

}

실행결과 arr1 : 004BFBE0

arr1+1 : 004BFBE8

arr1+2 : 004BFBF0 //arr1에서 +1마다 8씩 증가

arr2: 004BFBC0

arr2+1: 004BFBCC //arr2에서 +1 마다 12씩 증가

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| arr1 ----------->  ㄴ>arr1+1  ㄴ>arr1+2 | 1열 | 2열 | 1행  2행  3행 | arr2---------->  arr2+1----------> | 1열 | 2열 | 3열 |
| + | + | + | + | + |
|  |  |  |  |  |
|  |  |  |  |  |

2차원 배열의 포인터형은 가로의 길이에 따라서도 달라진다. 이것이 2차원 배열이름의 포인터형을 결정짓기 어렵게 하는 요소

@ 2차원 배열의 포인터형

Int arr[3][4];

배열 이름 arr이 가르키는 대상은 int형 변수이고, arr값을 1증가하면 실제로는 sizeof(int)x4의 크기만큼 주소값이 증가하는 포인터형

ㄴ> 이러한 유형의 포인터 변수 선언 방법

Int (\*ptr)[4]; //배열 포인터 변수 라고 함

Int(\*ptr)[4]; //ptr은 포인터

Int(\*ptr)[4]; //int형 변수를 가르키는 포인터

Int (\*ptr)[4]; //포인터 연산 시 4칸 건너뛰는 포인터

#include<stdio.h>

Int main(void){

Int arr1[2][2] = { { 1 , 2}, { 3 , 4 } };

Int arr2[3]2[] = { { 1 , 2 }, { 3 , 4 }, { 5 , 6 } }

Int (\*ptr)[2];

Int I;

Ptr = arr1;

Printf(“ \*\* show2, 2arr1 \*\* \n”);

For (i=0; i<2; i++){

Printf(“5d 5d \n”, ptr[i][0], ptr[i][1]); //포인터 형이 같으면 동일한 방식으로 접근 가능

}

Ptr = arr2;

Printf(“ \*\*show 3, 2 arr2 \*\* \n”);

For(i=0; i<3; i++){

Printf(“%d %d \n”, ptr[i][0], ptr[i][1]);

}

Return 0;

}

실행결과 \*\* show 2, 2 arr1 \*\*

1. 2
2. 4

\*\* show 3, 2 arr2 \*\*

1. 2
2. 4

5 6

@ **배열포인터** 와 **포인터배열** 의 차이

Int \*who[4]; //포인터 배열

Int (\*whoB)[4]; //배열 포인터

#include<stdio.h>

Int main(void{

Int num1 = 10, num2 = 20, num3 = 30;

Int arr2d[2][3] = { { 1, 2, 3 }, { 4, 5, 6 } };

Int I, j;

Int \*whoA[3] = { &num1. &num2, &num3 };

Int (\*whoB)[3] = arr2d;

Printf(“%d %d %d \n”, \*whoA[0], \*whoA[1], \*whoa[2]);

For ( i=0; i<2; i++{

For(j=0; j<3; j++)

Printf(“%d”, whoB[i][j]);

Printf(“\n”);

}

Return 0;

}

실행결과 10 20 30

1 2 3

4 5 6

@ 2차원 배열을 함수의 인자로 전달

Int main(void){

Int arr1[2][7];

Double arr2[4][5];

SimpleFunc(?, ?);

Void simpleFunc(int (\*parr1)[7], double (\*parr2)[5])

=

Void simpleFunc(int parr[ ][7], double parr2[ ][5])

// (\*parr1) = parr1[ ] 인것은 매개변수의 선언에서만 같은 의미

#include<stdio.h>

Void showArr2DStyle(int (\*arr)[4], intc){

Int I, j;

For (i=0; o<c; i++){

For(j=0; j<4; j++)

Printf(“%d”, arr[i][j];

Printf(“\n”);

}

Printf(“\n”);

}

Int main(void){

Int arr1[2][3] = { 1, 2, 3, 4, 5, 6, 7, 8 };

ShowArr2DStyle(arr1, sizeof(arr1)/sizeof(arr[0]));

Return 0;

}

실행결과 1 2 3 4

5 6 7 8

@2차원 배열에서도 arr[i] = \*(arr+i)

Int arr[3][2] = { { 1, 2, }, {3, 4}, {4, 5} };

Arr[2][1] = 4;

ㄴ>

(\*(arr+2))[1] = 4;

이 식은 arr[2][1] = 4에서 arr[2]를 \*(arr+2)로 바꾸면 됨

하지만 이 다음엔 치환이 필요함 arr[2][1] = 4에서 arr[2]를 A로 치환

ㄴ>

A[1] = 4 -> \*(A + 1) = 4

다음 A를 원래대로 되돌려보면

\*(arr[2]+1) = 4 - > \*(\*(arr+2)+1) = 4;