**포인터**

\*메모리에 직접 접근 가능

기본적인 틀

int main (void)

{

int num = 7;

int \*pnum;

pnum = &num;

}

int main(void)

{

int num = 10;

\*pnum = 20; // pnum이 가르키는 메모리 공간인 변수 num에 접근

printf(“%d”, \*pnum);

}

다양한 포인트형이 존재

int \*pnum //저장된 주소를 시작으로 4바이트 읽어들여 정수로 해석

double \*pnum // 저장된 주소를 시작으로 8바이트 읽어들여 실수로 해석

type\* 포인터의 형은 메모리 공간을 참조하는 기준이 됨

\*변수의 자료형에 맞지않는 포인터 변수의 선언

int main(void)

{

double num = 3.14;

int \*pnum = &num; => 두개가 불일치 하더라도 컴파일 에러는 발생하지않음

printf(“%d”,\*pnum); // 아무 의미없는 쓰레기값을 출력

}

잘못된 포인터의 사용

int \*ptr; //선언만하고 초기화 안하면 쓰레기값으로 초기화

\*ptr = 200; // 어디를 가르키는지도 모르는데 값을 저장하는경우 시스템 자체에 치명적인 문제를 일으킬 수 있음.

NULL

int main(void)

{

int \*ptr1=0;

int \*ptr2 =NULL; //NULL은 사실상 0을 의미함

}

이것이 의미하는 것은 0번지를 의미하는 것이 아니라, 아무데도 가르키지 않음 을 의미한다.

**포인터와 배열**

배열은 값을 바꿀 수 없는 상수 형태의 포인터 이다.

배열의 이름도 포인터 이기 때문에 \*연산 가능

#include <stdio.h>

int main(void)

{

int arr[3] = { 1, 2, 3 };

double arr2[3] = {1.1 , 2.2, 3.3};

printf(“%d %.1lf\n”, \*arr, \*arr2);

\*arr +=100;

\*arr2 =+120.5 🡺 가르키는 값에 값을 더하고있다.

printf(“%d %.1lf\n”,arr[0],arr2[0]);

return 0;

}

실행 결과 : 1 1.1

101 121.6

포인터를 배열의 이름으로 사용 가능

#include <stdio.h>

int main(void)

{

int arr[3] = {15 , 25, 35};

int \*ptr = &arr[0];

printf(“%d %d\n”, ptr[0],arr[0]);

printf(“%d %d\n”, ptr[1],arr[1]);

return 0;

}

실행 결과 : 15 15

25 25

\*포인터 연산

#include <stdio.h>

int main(void)

{

int arr[3] = {11 , 22, 33};

int \*ptr = arr;

printf(“%d %d %d\n”, \*ptr , \*(ptr+1), \*(ptr+2));

printf(“%d”,\*ptr); ptr++; //printf 호출 후 , ptr++ 실행

printf(“%d”, \*ptr); ptr++;

printf(“%d”,\*ptr); ptr--; // print 호출 후 ptr – 실행

printf(“%d”, \*ptr); ptr--;

printf(“%d”, \*ptr);

printf(“\n”);

return 0;

}

**두가지 연산의 차이점**

\*(++ptr) = 20; //ptr에 저장된 값 자체를 변경

\*(ptr+1) = 20; //ptr에 저장된 값 자체는 변경되지 않음

값 자체를 변경한다는건 맨 처음에 int\* ptr = &arr[0]; 이라고 줬을 때 ptr[0]은 arr[0]을 가르키게 됨

\*(++ptr)=20 은 하나 증가되어 ptr[0] 자체가 arr[1]로 넘어가니 ptr 자체에 값을 변경한다는 것

반면에 \*(ptr +1) 이나 \*(ptr+2) 같은건 계속 처음에 기준 잡은 ptr[0]은 arr[0]으로 잡히기 때문에 ptr에 저장된 값 자체는 변경되지 않는다.

**두가지 형태의 문자열**

변수 형태의 문자열

Char str1[] = “My String”; //배열의 길이는 자동으로 계산됨

Char str[] = “My String”;

str = “Your String”; //가르키는 대상 변경시 에러발생

str[0] = ‘A’ //문자열 변경 가능

상수형태의 문자열

char \* str2 = “My String”;

str2 = “Your String”; // 가르키는 대상 변경 가능

str2[0] = ‘A’; //문자열 변경시도시 컴파일 에러는 발생하지 않지만 생각했던 결과가 나오지 않음 , 컴파일러에 따라서 종료시키거나 연산을 무시해버린다.

포인터 배열

* 주소값의 저장이 가능한 배열

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

int \* arr[3]={ &num1, &num2, &num3 };

arr[0] = 10;

arr[1] = 20;

arr[2] = 30;

문자열을 저장하는 포인터 배열

char \* strArr[3] = { “simple” , “string” , “Array” };

strArr[0] = “simple”;

strArr[1] = “string”

strArr[2] = “Array”

포인터와 함수

* 매개변수로는 배열을 선언 할 수 없다.
* 대신 함수내에서 배열에 접근 할 수 있도록 배열의 주소 값 전달 가능

배열을 함수의 인자로 전달하는 방식

int arr[3] = { 1, 2, 3 };

simpleFunc (arr); //simpleFunc 함수 호출하면서 arr 주소 전달

* arr -> 매개변수 선언 방법

int main(void)

{

int arr[3] = { 1, 2, 3 };

int \*ptr = arr;

}

Void simpleFunc(int \*param)

{

~~

}

#include <stdio.h>

void showArrayElem(int \*param , int len)

{

int i;

for(i =0; i<len; i++)

{

printf(“%d”, param[i]); //배열을 참조

}

Printf(“\n”);

}

void AddArrayElem(int \*param, int len, int add)

{

int i;

for(i=0; i<len; i++)

{

Param[i] +=add; //접근하여 저장된 값 참조 및 변경

}

}

int main(void)

{

int arr[3] = { 1, 2, 3 };

AddArrayElem(arr , sizeof(arr) / sizeof(int) ,1);

showArrayElem(arr, sizeof(arr)/sizeof(int));

return 0;

}

배열을 함수의 인자로 받는 또 다른 선언 방법

void Func(int \*param) // 형태 완전히 동일하게 선언

void Func(int param[ ]) //하지만 이둘이 같은 선언으로 간주되는 경우는 매개변수 선언으로 제한 됨

int main(void)

{

int arr[3] = { 1, 2, 3 };

int \*ptr = arr; // int ptr [ ] = arr; 로 대체 불가능

}

값을 전달하는 형태의 함수호출 : call by value

* 함수 호출방식을 구별하는 것 에는 큰 의미는 없지만, 아래와 같은 실수를 막기 위함임

#include <stdio.h>

void swap(int n1, int n2)

{

int temp = n1;

n1 = n2;

n2 = temp;

printf(“n1 n2 : %d %d \n”, n1, n2);

}

int main(void)

{

int num1 = 10;

int num2 = 20;

printf(“num1 num2 : %d %d \n”,num1 ,num2);

swap(num1 , num2);

printf(“num1 num2 : %d %d \n” , num1 ,num2);

return 0;

}

실행 결과 : num1 num2 : 10 20

n1 n2 : 20 10

num1 num2 “ 10 20

main 함수 영역 swap함수 영역

num1 10 🡺 n1 : 10 (n1 <-> n2)

num2 20 🡺 n2 : 20 (n1 <-> n2)

주소값을 전달하는 형태 함수 호출 : call by reference

#include <stdio.h>

void swap(int \*ptr1, int\*ptr2)

{

int temp = \*ptr1;

\*ptr1 = \*ptr2;

\*ptr2 = temp;

}

int main(void)

{

int num1 = 10;

int num2 = 20;

printf(“num1 num2 : %d %d \n”, num1 ,num2 );

swap(&num1, &num2);

printf(“num1 num2 : %d %d \n”,num1 ,num2);

return 0;

}

실행 결과 : num1 num2 : 10 20

num1 num2 : 20 10

num1 -> temp -> num2 -> num1

포인터 문제

1. 배열내에서 첫번째로 큰 값과 두번째로 큰 값 찾기 (call-by-reference)
2. 정수 배열 A[ ] 를 정수 배열 B[ ]에 복수하는 함수를 작성하고 출력하라
3. 5개의 알파벳을 입력받는 배열을 만들고 입력받은 알파벳이 소문자일 경우 대문자 출력 , 대문자일 경우 소문자 출력
4. 세 변수에 저장된 값을 서로 뒤 바꾸는 함수를 정의해보자

num1 값은 num2 에 num2값은 num3에 num3값은 num1애 저장되어야함