포인터

=> 메모리의 주소를 가지는 변수.

실행중인 프로세스의 영역의 주소.

포인터 변수

=> 다른 변수의 메모리주소를 저장하는 변수.

=> 형식 : 자료형 * 변수이름;

ex) int* ptr; ← 정수형 포인터 변수

float* ptr; ← 실수형 포인터 변수

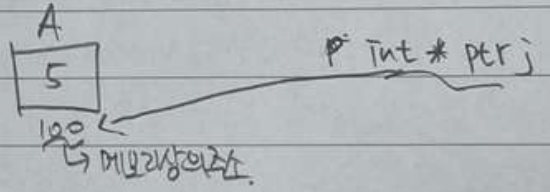
char* ptr; ← 문자형 포인터 변수.

64비트 버전에서 메모리 내의 주소는 8바이트이다.

=> 포인터가 가르키는 것은 메모리 주소이다.

ex) int a = 5;

int* ptr;



=> 포인터의 주소를 선언할때는 &를사용한다.

ex) &a => a변수의 주소

해당주소의 값을 확인하는법

'*'사용하기

ex) int a = 5; => 변수선언

int* ptr; => 포인터 변수선언

ptr = &a; => a 변수의 주소(a의주소를 ptr에게 알려줌)

printf("value of ptr is %d", *ptr);

↳ a의주소에 있는 실제값

Date. No.

함수의 구조.

ex) void func1(void))=> 함수의 헤더(함수의 형태)

{

printf("I ");)=> 함수의 바디(함수 실행부)

}

=> (자료타입) (함수의 이름) (인자/variable)

ex) void star (int a) {

if (a==1) {

printf ("안녕하세요\n")

}(a==2)

if(a==2) {

printf(" 반갑습니다\n")

}

int main() {

printf ("hello world\n");

star(1);

star(2);

=) 출력.

hello world

안녕하세요

반갑습니다.

함수란?

⇒ 어떤 기능을 수행하는 독립적인 실행단위

⇒ 복잡하거나 앞으로자주 반복적으로 쓰인것같은 행동이나 동작이 있다면 함수로 만들어서 그 행동을 할 때마다 함수를 사용.

ex) 우리가 자주 쓰는 printf() 도 실제코드는 훨씬 복잡하게 설계됨.

함수를 만들때 주의할점

⇒ 하나의함수에 모든기능을 기술하지 않고, 각각은 독립적인 함수에 분리하여 기술하기

∴ 예를들어 은행계좌를 확인하는 모든기능을 담은 하나의 함수를 만든다고생각해보자

그럼만약 그함수는

```
main(){
  본인이름 입력코드
  본인이름 출력코드
  계좌번호 입력코드
  계좌번호 출력코드
  :
}
```

이런형식으로 이루어 질것인데

이런식으로 함수를 구성하면,

- 함수의크기가 너무커지고, 실행속도가느려짐
- 가독성이 떨어짐
- 오류가있을때 수정이어려움
- 필요할때 호출할수 없음

:

등의 문제가있다.

따라서. 하나의 함수에 모든기능을 기술하지않고

각각의 독립적인 기능을하는 함수를 분리하여 기술하게됨

기능별로 함수를 나누어 기술해야하는이유

⇒ 함수를 기능별로 나누어 기술하게되면 함수자체의 가독성.

만들어놓은 함수는 호출이 가능.

가독성 향상, 에러검출에 이점.

함수의 종류

⇒ 내가 만들함수 ex) printf(), scanf(), for() . .

내가 만든 함수

형변환

⇒ 하나의 자료형을 다른자료형으로 변환시켜 자료형을 같게하는 것

변수 2개로 연산을 수행할경우 사용

묵시적 형변환

⇒ 컴파일러가 자동으로 변환

데이터의 값을 잃지 않는 쪽으로 형변환이 이루어짐

char ⇒ short ⇒ int ⇒ unsigned ⇒ long ⇒ float ⇒ double

→ 형변환이 일어나는 방향.

ex) a가 정수형, b가 실수형인 경우

```
int a= 5
float b=10.5, c;
c= a+b;
```

⇒ 이런식으로 연산을 할때 a는 실수형으로 자동으로 형변환이 일어나 a=5.00 이된다.

명시적 형변환

⇒ 사용자가 강제로 형변환을 시키는것

연산의 앞에 ()를 붙이고, () 안에는 변환시키려는 자료형 기록.

ex) float형 변수 b를 int형으로 강제변환

⇒ float a = 7.55

float b = 13.57; c;

c= a+(int)b; → (int)b = 13

printf(" c= a+(int)b 의 결과는 %.2f 이다" f)

⇒ c= a+(int)b의 결과는 20.55 이다

int a=5;
int b=10;) ⇒ 변수선언.

int* ptr;
int* ptr2;) ⇒ 포인터선언

ptr = &a;
ptr2 = &b;) ⇒ 포인터에 a,b의주소를알려줌

printf("b-a = %d", *ptr2 - *ptr);

↳ (ptr2 에있는 실제값) - (ptr 에있는 실제값) ⇒ 10-5=5

⇒ 포인터는 +, -, ++, -- 를이용해 연산이 가능하다.

배열과 포인터.

⇒ 배열은 포인터를 이용해 제어가능.

⇒ char arr[6] = {"hello"} 라는 배열이 선언되었다고하면.

배열의 이름인 arr 을 포인터에 대입오해준다

char* prt;

prt = &arr[0]; ⇒ 배열시작주소대입

⇒ char arr[6] = {"hello"}

char* ptr;

ptr = &arr[0];

printf("%c", *ptr); ⇒ h.

주의해야할점

포인터는 메모리 부분에서 직접적으로 메모리에 저장되어있는 중요한정보들을 건드릴수있다

오류 메가 나면 어디가 어떻게 틀렸는지 파악이 힘들다.