포인터

|  |
| --- |
| 1.다음 설명 중 틀린 것은?  (1)메모리에 올려진 데이터를 제어하려면 오직 변수만을 이용할 수 있다.  (2)변수에 의한 방법이 아닌, 메모리 주소를 통한 접근으로도 데이터를 변경할 수 있다.  (3)메모리를 접근할때 사용되는 연산자는 &이다.  (4)메모리에 접근할 때마다 &연산을 수행하는 것보다는, 구해진 주소값을 어딘가에 담아두고  사용하면 훨씬 편할 것이다.  (5)일반 변수에는 데이터를 담지만, 포인터 변수엔 주소값을 담는다.  (6)포인터도 변수이므로 메모리에 생성될때 크기를 가지며, 이 크기는 포인터가 가리키는 데이터  의 자료형이 char형일 경우 1byte 의 크기를 갖게 된다.  (7)포인터도 변수이므로, 자료형을 가지며 포인터가 가리키는 데이터의 자료형에 좌우된다. |
| 2.알맞는 연산자를 채워넣으세요.     |  |  | | --- | --- | | **연산자 / 기호** | **설명** | | & | 특정 변수의 주소값을 구하여 그 시작 주소값을 반환한다.  또한 그 **변수의 자료형 정보**도 내부적으로 구분한다. | | \* | 포인터 변수 선언시 사용되는 기호 | | \* | 포인터 변수가 가리키는 변수의 값을 참조하기 위한 연산자 | |
| 3.메모리에 int 형 변수 a=5를 선언하고 변수명을 이용하는 방법과 포인터를 이용하는 방법을  이용하여 데이터를 9로 바꿔보자 |
| 4.아래 코드에 대한 설명으로 틀린 것은?  int a = 3;  printf("%u", &a);  int \* pt = &a;  char \* ch = "abc";  (1) 메모리에 올라온 a변수의 주소값이 200으로 출력되었다면 a가 차지하는 메모리의 주소값의  시작값이 200인 것이다.  (2) 메모리에 올라온 a변수의 주소값이 200으로 출력되었다면 a가 차지하는 메모리 공간의  크기는 200부터 203까지로 예측이 가능하다.  (3) pt 변수는 a의 주소값을 담을 수 있는 포인터 변수이다.  (4) pt 포인터 변수의 크기는 ch 포인터 변수의 크기보다 4배 크다 |
| 5.아래의 실행 결과는 ?  int x = 7;  int \*pt = &x;  \*pt += 3;  \*&x = 7;  printf("%d", x); 7 |
| 6.다음 설명 중 틀린 것은?  (1) \*(간접 참조 연산자)를 사용하여 데이터를 변경하는 방식은, 변수를 이용하는 방식과 결과적  으로 같은 결과를 가져온다.  (2) 변수에 의한 방식보다 \* 연산자를 사용하는 편이 성능면에서 더 좋다  (3) 메모리에 올라온 데이터의 주소값을 알아내기 위해 &연산자를 사용해야 한다. 하지만 매번  연산을 수행하여 얻기 보다는 , 이미 얻어진 주소값 정보를 자주 사용하기 위해 변수에 담아  놓으면 개발시 훨씬 효율성이 있을 것이다.  (4) (3)의 이유로 지원되는 변수가 바로 포인터 변수이며, 포인터 변수는 주소값을 담아놓는 목  적으로 사용되어 진다. |
| 7.아래 코드의 실행 결과는 ?  char c = 'A';  short s = 100;  int x = 5;  long k = 1000;      char \*pt1=&c;  short \*pt2 = &s;  int \*pt3 = &x;  long \*pt4 = &k;  printf("%d, %d, %d, %d",sizeof(pt1), sizeof(pt2), sizeof(pt3), sizeof(pt4));  4byte |
| 8.다음 설명 중 틀린 것은?  int a = 3; (가)  int \*p = &a; (나)  \*p = 25; (다)  p = 8; (라)  printf("%d", a); (마)  (1) 결국 (다)와 같이 간접 참조 연산자를 이용하여 데이터를 제어하는 것은 변수 a를 이용하여  값을 제어하는 것과 같다.  (2) (나)와 (다) 에 사용된 \*기호는 모두 간접 참조 연산자라고 보면 된다.  (3) (라)에서 a변수가 담고 있던 데이터는 8로 변경된다.  (4) (마)의 결과는 3이 출력된다. |
| 9. 두수의 a, b 가 다음과 같을때, 값을 바꿔 출력하는 프로그램을 작성하시오.  int a= 10;  int b=20; |
| 10.아래의 코드에 대한 설명 중 틀린 것을 고르면?  int arr[5] = { 10, 20, 30, 40, 50 };  int \*pt = arr; //(가)  printf("%d \n", \*(arr + 1)); //(나)20  printf("%d \n", \*(pt+1));//(다)20  printf("%d \n", \*pt +1);//(라)11  (1) C에서 배열을 생성하면, 컴파일러에 의해 배열의 이름이 배열 요소의 첫번째 주소값으로 대  치 된다. 따라서 (가)에서 배열 첫번째 요소인 1의 주소값이 pt포인터 변수에 대입되므로,  결국 pt는 arr배열을 가리키고 있다고 볼 수 있다.  (2) (나)에서 arr+1 = 21이 되므로, 출력 결과는 21이 된다.  (3) (다)의 출력 결과는 20이다.  (4) (라)의 출력 결과는 20이다.  (5) 주소값에 정수를 더하면, 그 연산은 일반 데이터 연산과는 틀린 주소연산으로 처리된다.  따라서 주소값 +1 은 다음 해당 자료형 크기만큼 증가된 다음 주소값을 의미하게 된다. |
| 11.다음 설명 중 틀린 것은?  int arr[5] = { 1, 2, 3, 4, 5 };    (1) 배열의 길이가 5인 int 형 배열이 메모리에 올라왔을때를 표현한 것이다.  (2) 배열의 시작 요소의 주소값이 100이라는 사실만 알아도 나머지 요소들의 주소값은 충분히  예측이 가능하다.  (3) 배열명이 결국 주소값을 의미한다면, 주소값을 담을 수 있는 포인터 변수에 배열의 주소값을  담아서, 배열을 처리할 수 있는 것은 당연한 것이다.  (4) 위 (3)번의 설명이 맞다면 아래의 코드는 문제가 없을 것이다.  int \*p = arr;  (5) 배열의 각 요소를 출력하기 위한 아래의 코드는 문제가 없다.  **Arr[n] = arr + n;**  printf("%d, %d \n", \*(pt+0), pt[0]);  printf("%d, %d \n", \*(pt + 1), pt[1]);  printf("%d, %d \n", \*(pt + 2), pt[2]);  printf("%d, %d \n", \*(pt + 3), pt[3]);  printf("%d, %d \n", \*(pt + 4), pt[4]); |
| 12.아래 코드의 출력 결과는 ? **0,1,2**  int arr[3] = { 1, 2, 3 };  for (int i = 0; i < 3;i++) {  \*(arr + i) = i;  printf("%d", arr[i]);  } |
| 13.좋아하는 과일 10개의 이름을 문자열에 담아 놓고 포인터를 이용하여 출력하세요 |
| 14.다음 설명 중 틀린 것은?  (1) 변수는 메모리에 저장되지만, 함수는 메모리에 저장되지 않는다.  (2) 함수도 메모리에 저장되어 지므로, 주소값을 가지고 있으며 이 주소값을 담은 포인터를 가리  켜 함수 포인터라 할 수 있다.  (3) 배열명이 첫번째 요소의 주소값으로 대치되듯, 함수명도 메모리에 저장된 함수의 주소값으로  대치된다.  (4) 함수명은 얼마든지 다른 함수를 가리킬 수 있는 변수로 보아야 한다.  (5) 포인터 선언시 자료형을 명시하지 않으면 에러가 발생한다. |
| 15.다음과 같이 정의된 함수의 포인터를 선언하세요     |  |  | | --- | --- | | 정의된 함수 | 함수 포인터 | | double yourFunc(int a) {  } | Double (\*pp)(int) | | char myFunc(int a, int b) {  } | Char (\*pp)(int,int) | | void theyFunc() {  } | Void (\*pp)(void) | |
| 16.아래 코드(가)를 출력시키기 위해 (나)코드를 완성하세요  #include <stdio.h>  double yourFunc(int a) {  printf("yourFunc 호출했어요 %d", a); //(가)  }  void funcPointer(double(\*fp) (int)) {  fp(3);  }  int main() {  funcPointer(yourFunc); //(나)  return 0;  } |