자료구조 보고서 2주차

학과 : 소프트웨어학과

학번 : 2023041012

이름 : 김태영

`

{소스코드1}

git hub url :

https://github.com/Hoogdle/homework2/blob/main/2%EC%A3%BC%EC%B0%A8/lab1.c

```c
#include <stdio.h>
int main(){

 char charType; //문자형 데이터 변수 charType
 int integerType; //정수형 데이터 변수 integerType
 float floatType; //실수형 데이터 변수 floatType
 double doubleType; //double형 데이터 변수 doubleTy
 printf("--------[%s] [%s]--------\n","김태영","2023041012");
 printf("Size of char: %ld byte\n",sizeof(charType)); //charType 변수의 사이즈 확인, 문자형이므로 1바이트
 printf("Size of int: %ld bytes\n",sizeof(integerType)); // integerType 변수의 사이즈 확인, int형이므로 4바이트
 printf("Size of float: %ld bytes\n",sizeof(floatType)); // floatType 변수의 사이즈 확인 float형이므로 4바이트
 printf("Size of double: %ld bytes\n",sizeof(doubleType)); // doubleType 변수의 사이즈 확인 double형이므로 8바이트
 printf("----------------------------------------\n");
 printf("Size of char: %ld byte\n",sizeof(char)); // 자료형 char의 사이즈 확인, char은 1바이트 크기를 가지므로 1바이트
 printf("Size of int: %ld bytes\n",sizeof(int)); // 자료형 int의 사이즈 확인, int은 4바이트 크기를 가지므로 4바이트
 printf("Size of float: %ld bytes\n",sizeof(float)); // 자료형 float 사이즈 확인, float은 4바이트 크기를 가지므로 4바이트
 printf("Size of double: %ld bytes\n",sizeof(double)); // 자료형 double 사이즈 확인, double은 8바이트 크기를 가지므로 8바이트
 printf("----------------------------------------\n");
 // cf) pc가 64비트 머신이라 포인터의 크기는 8바이트이지만 vscode가 32비트 기준으로 컴파일을하여 출력 결과는 4바이트가 나옵니다.
 printf("Size of char*: %ld byte\n",sizeof(char*)); // char을 가리키는 포인터의 크기를 확인합니다. 포인터는 주소를 가리키기 때문에 가리키는 데이터 타입과 상관없이 8바이트의 크기를 가집니다.
 printf("Size of int*: %ld bytes\n",sizeof(int*)); // int를 가리키는 포인터의 크기를 확인합니다. 포인터는 주소를 가리키기 때문에 가리키는 데이터 타입과 상관없이 8바이트의 크기를 가집니다.
 printf("Size of float*: %ld bytes\n",sizeof(float*)); // float를 가리키는 포인터의 크기를 확인합니다. 포인터는 주소를 가리키기 때문에 가리키는 데이터 타입과 상관없이 8바이트의 크기를 가집니다.
 printf("Size of double*: %ld bytes\n",sizeof(double*)); // double를 가리키는 포인터의 크기를 확인합니다. 포인터는 주소를 가리키기 때문에 가리키는 데이터 타입과 상관없이 8바이트의 크기를 가집니다.
 return 0;
}
```

```
--------[김태영] [2023041012]--------
Size of char: 1 byte
Size of int: 4 bytes
Size of float: 4 bytes
Size of double: 8 bytes
----------------------------------------
Size of char: 1 byte
Size of int: 4 bytes
Size of float: 4 bytes
Size of double: 8 bytes
----------------------------------------
Size of char*: 4 byte
Size of int*: 4 bytes
Size of float*: 4 bytes
Size of double*: 4 bytes
 *  Terminal will be reused by tasks, press any key to close it.
```

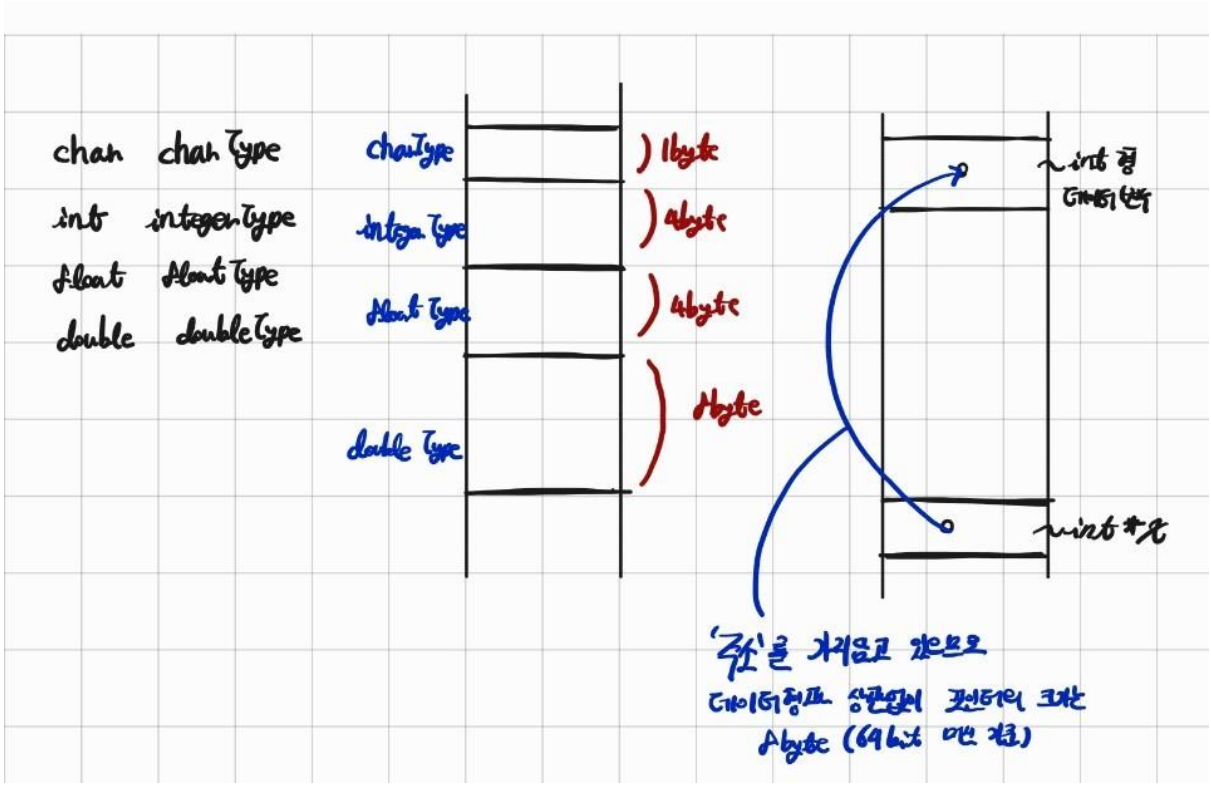
(*64비트 머신 기준, 포인터의 크기가 8바이트 지만 vscode가 자체적으로 32비트 머신으로 컴파일 하여 포인터의 크기가 4바이트로 출력되었습니다..)

vscode와 gcc 컴파일러로 코드를 작성 및 컴파일 하였습니다. 각 코드 옆에 주석으로 해당 코드가 어떤 작업을 수행하는지 작성하였습니다.

위 과정을 통해 자료형과 해당 자료형으로 선언된 변수의 크기와 포인터 변수의 크기를 확인 했습니다.

`

위 코드에 대한 모식도는 아래와 같습니다.



{소스코드2}

git hub url :

https://github.com/Hoogdle/homework2/blob/main/2%EC%A3%BC%EC%B0%A8/lab2.c

자료구조 과제 > 2주차 > C lab2.c > main()

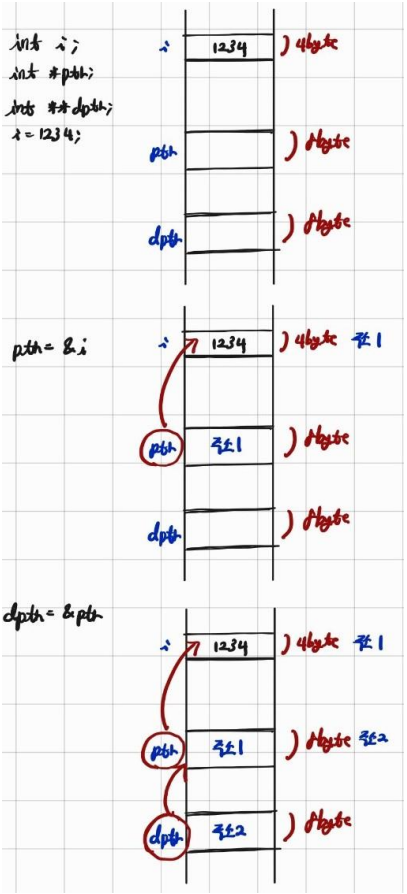
```c
#include <stdio.h>
int main()
{
    int i;
    int *ptr;
    int **dptr;
    i = 1234;

    printf("--------[%s] [%s]--------\n","김태영","2023041012");
    printf("[checking values before ptr = &i] \n");
    printf("value of i == %d\n", i); // i의 값 확인
    printf("address of i == %p\n", &i); // i의 주소 확인
    printf("value of ptr == %p\n", ptr); // # ptr이 가리키는 변수의 주소 확인
    printf("address of ptr == %p\n", &ptr); // ptr(자체) 주소 확인
    ptr = &i; /* ptr is now holding the address of i */ //ptr이 i의 주소를 가리키도록 초기화
    printf("\n[checking values after ptr = &i] \n");
    printf("value of i == %d\n", i); // i의 값 확인
    printf("address of i == %p\n", &i); // i의 주소 확인
    printf("value of ptr == %p\n", ptr); // ptr이 가리키는 변수(i)의 주소 확인
    printf("address of ptr == %p\n", &ptr); // ptr 자체의 주소 확인
    printf("value of *ptr == %d\n", *ptr); // ptr이 가리키는 변수(i)의 값 확인
    dptr = &ptr; /* dptr is now holding the address of ptr */ // 이중 포인터 dptr이 포인터 ptr를 가리키도록 초기화
    printf("\n[checking values after dptr = &ptr] \n");
    printf("value of i == %d\n", i); // i의 값 확인
    printf("address of i == %p\n", &i); // i의 주소 확인
    printf("value of ptr == %p\n", ptr); //ptr이 가리키는 변수(i)의 주소 확인
    printf("address of ptr == %p\n", &ptr); //ptr 자체의 주소 확인
    printf("value of *ptr == %d\n", *ptr); //ptr이 가리키는 변수(i)의 값 확인
    printf("value of dptr == %p\n", dptr); // dptr이 가리키는 변수(ptr)의 주소 확인
    printf("address of dptr == %p\n", &dptr); // dptr 자체의 주소 확인
    printf("value of *dptr == %p\n", *dptr); // dptr이 가리키는 변수(ptr)의 값(i의 주소) 확인
    printf("value of **dptr == %d\n", **dptr); // dptr이 가리키는 변수(ptr)가 가리키는 변수(i)의 값 확인
    *ptr = 7777; /* changing the value of *ptr */ // ptr이 가리키는 변수(i)의 값을 7777로 초기화
    printf("\n[after *ptr = 7777] \n");
    printf("value of i == %d\n", i); // i의 값 확인
    printf("value of *ptr == %d\n", *ptr); // ptr이 가리키는 변수(i)의 값 확인
    printf("value of **dptr == %d\n", **dptr); // dptr이 가리키는 변수(ptr)이 가리키는 변수(i)의 값 확인
    **dptr = 8888; /* changing the value of **dptr */ // dptr이 가리키는 변수(ptr)가 가리키는 변수(i)의 값을 8888로 초기화
    printf("\n[after **dptr = 8888] \n");
    printf("value of i == %d\n", i); // i의 값 확인
    printf("value of *ptr == %d\n", *ptr); // ptr이 가리키는 변수(i)의 값 확인
    printf("value of **dptr == %d\n", **dptr); // dptr이 가리키는 변수(ptr)가 가리키는 변수(i)의 값 확인
    return 0;
}
```

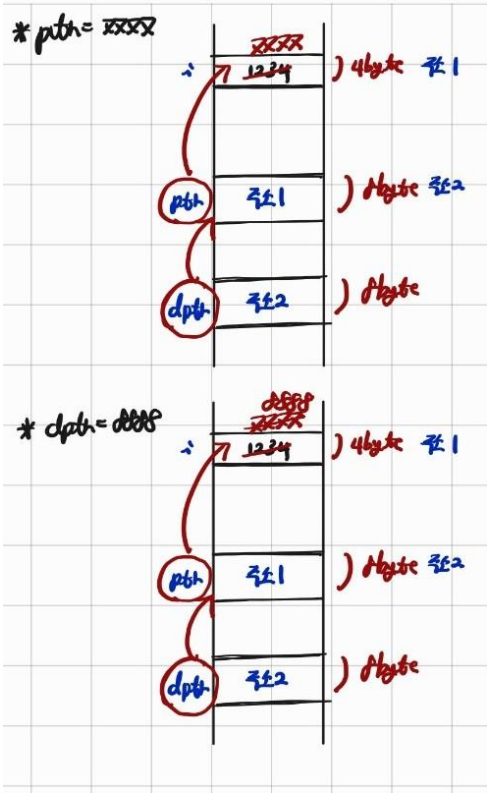
```
--------[김태영] [2023041012]--------
[checking values before ptr = &i]
value of i == 1234
address of i == 0061FF1C
value of ptr == 00280000
address of ptr == 0061FF18

[checking values after ptr = &i]
value of i == 1234
address of i == 0061FF1C
value of ptr == 0061FF1C
address of ptr == 0061FF18
value of *ptr == 1234

[checking values after dptr = &ptr]
value of i == 1234
address of i == 0061FF1C
value of ptr == 0061FF1C
address of ptr == 0061FF18
value of *ptr == 1234
value of dptr == 0061FF18
address of dptr == 0061FF14
value of *dptr == 0061FF1C
value of **dptr == 1234

[after *ptr = 7777]
value of i == 7777
value of *ptr == 7777
value of **dptr == 7777

[after **dptr = 8888]
value of i == 8888
value of *ptr == 8888
value of **dptr == 8888
 *  Terminal will be reused by tasks, press any key to close it.
```

vscode와 gcc 컴파일러로 코드를 작성 및 컴파일 하였습니다. 각 코드 옆에 주석으로 해당 코드가 어떤 작업을 수행하는지 작성하였습니다.

위 과정을 통해 변수와 포인터의 관계, 포인터와 이중 포인터의 관계를 정리할 수 있었습니다.

위 코드에 대한 모식도는 아래와 같습니다.





`