1) 메모리 주소

16진수

메모리의 세계에서는 16 진수를 사용한다.
0123456789ABCDEF
ex)10 진수 - 30 -> 16 진수 - 1E
16 진수는 앞에 0x 를 붙인다. 그래서 0x1E

메모리 주소

만약, 정수 50 을 저장하면 int 이므로 메모리 어딘가에서 4 바이트를 차지하고 있을 것이다.

```
#include <stdio.h>


int main(void)

{

	int n = 50;

	printf("%i₩n", n);

}
```

을 입력하면 50 이 나온다.

n이 메모리에서 어디 저장되어있나 보고싶을 때 (메모리 주소)

```
#include <stdio.h>
```

```
int main(void)

{

	int n = 50;

	printf("%p₩n", &n);

}
```

%i 대신 %p, n 대신 &n 을 입력하면 메모리 주소인 0x7ffe00b3adbc 가 나오게 된다.

이때, %i, 그리고 *&n 을 넣어주면

```
#include <stdio.h>



int main(void)

{

	int n = 50;

	printf("%i₩n", *&n);

}
```

n 의 메모리 주소를 얻고, 다시 그 주소에 해당되는 값인 50 을 출력하게 된다. 결국 50 이라는 값만 나오는 것이다.

2)포인터

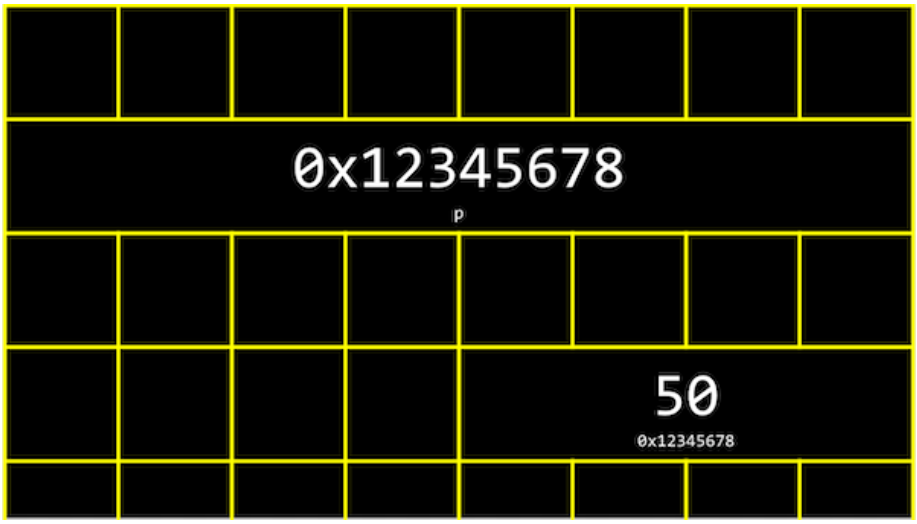
```
#include <stdio.h>
```

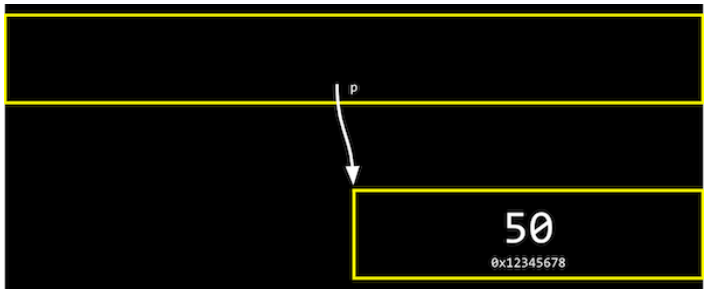
```
int main(void)

{

    int n = 50;

    int *p = &n;

    printf("%p₩n", p);

    printf("%i₩n", *p);

}
```

*은 메모리 주소에 있는 값을 받아온다. 이 연산자를 이용해 포인터 역할을 하는 변수를 선언할 수 있다.
*p 라는 포인터 변수에 &n 이라는 값, 즉 변수 n 의 주소를 저장한다.

첫 번째 printf 문과 같이 포인터 p 의 값, 즉 변수 **n 의 주소를 출력**하거나,
두 번째 printf 문과 같이 포인터 **p 가 가리키는 변수의 값**, 즉 변수 n 의 값을 출력할 수 있다.

잘 이해가 안가는데, 그림으로 보면





포인터의 크기는 메모리에 크기와 비례한다.

3) 문자열

string 과 char *
우리가 그동안 써왔던 string 은 cs50 에서 만든 것으로, 보조바퀴였다.

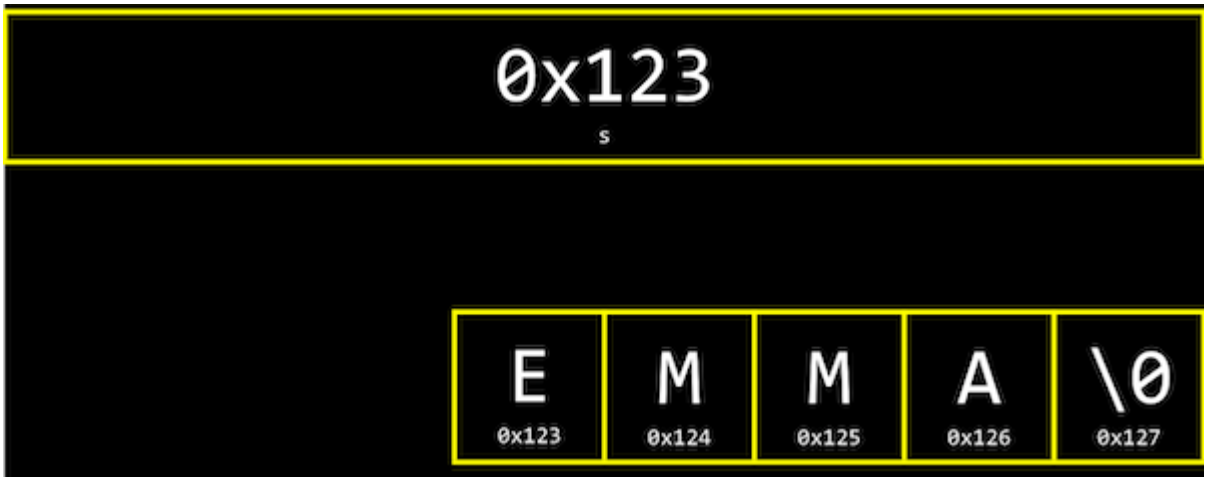
```
#include <cs50.h>

#include <stdio.h>



int main(void)

{

    string s = "EMMA";

    printf("%s₩n", s);

}
```

실은 char * 을 써야했다.

```
#include <stdio.h>
```

```
int main(void)

{

	char *s = "EMMA";

	printf("%s₩n", s);

}
```

char *s 에서 s 라는 변수는 문자에 대한 포인터이며, "EMMA"라는 문자열의 가장 첫 번째 값을 저장한다.



printf("%s₩n, s); 대신 printf("%p₩n, s); 을 쓰면
메모리의 주소가 나온다.

```
#include <stdio.h>


int main(void)

{
```

```
    char *s = "EMMA";
    printf("%p₩n", s);
}
```

s 는 문자열이므로 단어 각각의 주소를 알고싶으면

```
    printf("%p₩n", &s[0]);
    printf("%p₩n", &s[1]);
    printf("%p₩n", &s[2]);
    printf("%p₩n", &s[3]);
    printf("%p₩n", &s[4]);
```

4) 문자열비교

*s = s 가 가리키는 곳으로 찾아가면 뭐가 있을까?

```
printf("%c₩n", *s);
printf("%c₩n", *(s+1));
printf("%c₩n", *(s+2));
printf("%c₩n", *(s+3));
```

E (S 는 문자열 첫 글자의 주소가 나온다)
M
M
A

가 나오게 된다.

문자열비교

```
#include <cs50.h>

#include <stdio.h>


int main(void)

{

    // 사용자로부터 s와 t 두 개의 문자열 입력받아 저장

    string s = get_string("s: ");

    string t = get_string("t: ");


    // 두 문자열을 비교 (각 문자들을 비교)

    if (s == t)

    {

        printf("Same₩n");

    }

    else

    {

        printf("Different₩n");

    }

}
```

s: EMMA t:EMMA => Different 가 출력된다.
그 이유는 s 와 t 에는 메모리 주소가 적혀져 있는데
s 와 t 의 메모리 주소가 다르기 때문이다.
Same 이라는 결과가 나오기 위해서는 실제 문자열이 저장된 곳으로 이동해야 한다.

5) 문자열 복사

문자열을 복사하기 위해 다음과 같은 코드를 짠다면

```
#include <cs50.h>

#include <ctype.h>

#include <stdio.h>



int main(void)

{

    string s = get_string("s: ");

    string t = s;



    t[0] = toupper(t[0]); //대문자로 바꾸는 함수



    printf("s: %s₩n", s);

    printf("t: %s₩n", t);

}
```

s: emma 일 때, s 와 t 모두 Emma 가 된다.

그 이유는 string t = s 라고 했을 때, emma 의 메모리 자리가 그대로 t 에도 적용되기 때문이다.

이를 해결하기 위해서는, t 에 메모리를 할당해줘야 한다.

```
#include <cs50.h>

#include <ctype.h>

#include <stdio.h>

#include <string.h>


int main(void)

{

    char *s = get_string("s: ");

    char *t = malloc(strlen(s) + 1); //malloc 은 메모리 할당 함수


    for (int i = 0, n = strlen(s); i < n + 1; i++)

    {

        t[i] = s[i];

    }

    t[0] = toupper(t[0]);

    printf("s: %s₩n", s);

    printf("t: %s₩n", t);

}
```

이렇게 하면, s 의 문자열에 있는 문자 하나하나를 t 로 복사하게 된다.
+1 을 한 이유는 ₩n 의 자리를 만들어 주기 위함이다.