2018. 2. 28 수 – 6 회차
1~3. <stdlib.h>에서 사용
1. typedof
#def 와 비슷한 기능
자료형에 새로운 이름을 부여할 때 사용(주로 함수, 구조체에 사용됨)

Ex>
// Typedef 은 type 을 치환
// #define 은 type 외의 모든 것을 치환(ex. 함수)

```
#include <stdio.h>

typedef int INT[5];

int main(void)
{
   int i;
   INT arr = {1, 2, 3, 4, 5};

   for(i = 0; i < 5; i++)
      printf("arr[%d] = %d\n", i, arr[i]);

   return 0;
}
```

2. malloc()
Memory 구조상 Heap 에 data 를 할당함
data 가 계속 들어올 경우, 얼만큼의 data 가 들어오는지 알수가 없음
들어올 때마다 동적으로 할당할 필요성이 있음

Ex>
```
/*
2. malloc() - 네트워크분야에 필수적 : 동적인 움직임이 있으면 필요
주로 사용하는 분야 : 게임   그 외 : 도청기, 네트워크장비(라우터, 스위치)

게임에 많이 쓰이는 이유 : 게임이 새로 출시할 경우, 몇 명의 게임에 접속할 지 감이 안옴. 그러므로 미리 배열을 짜놓기가 무리
위의 상황을 처리하기 위하여 malloc() 사용

단점 : 속도가 느림(ex. 과거 중국인들이 우리나라 게임서버를 공격하기 위하여 단체로 접속했다가 나갔다를 반복, 결국 heap 이 그 속도를 따라가지 못하여 서버가 뻗어버림)
*/

/*
3. free() Function   -  malloc()과 한 쌍
Memory 구조상 heap 에 data 를 할당해제함
malloc()의 반대 역할을 수행함
free 도 malloc 처럼 자주하면 안좋음
*/
```

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

int main(void)
{
  char *str_ptr = (char*)malloc(sizeof(char) * 20);            //사이즈가 20 짜리

  printf("Input String:");
  scanf("%s", str_ptr);

  if(str_ptr != NULL)                                  //NULL 이 들어가므로 실제로는 19 개가 들어감
    printf("string = %s\n", str_ptr);

  free(str_ptr);

  return 0;
}
```

3. calloc()
malloc()과 같은 기능
차이점 :
malloc() - sizeof()를 정해줌
calloc() - 개수를 정해줌

Ex>
// stack 에 함수를 저장지 함수가 종료되면 더 이상 사용하지못함, 그러나 heap 에 저장하면 함수가 종료되도 주소만 알면 다시 사용가능(단, main 은 살아있어야한다)

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

int main(void)
{
  int *num_ptr = (int *)calloc(2, sizeof(int));

  printf("Input Integer : ");
  scanf("%d%d", &num_ptr[0], &num_ptr[1]);

  if(num_ptr != NULL)
    printf("Integer = %d, %d\n", num_ptr[0], num_ptr[1]);

  free(num_ptr);

  return 0;
}
```

4. construct
구조체

Ex>
/*
1. Understanding struct
구조체는 왜 사용하는가?
자료를 처리하다보니 하나로 묶어야 편함
"문자열과 숫자" 를 한 번에 묶어서 관리하고 싶을때등

구조체 - custom data type : 내가 원하는 데이터타입을 만들수있다

*/

```
#include <stdio.h>

struct pos
{
  double x_pos;
  double y_pos;
};

int main(void)
{
  double num;
  struct pos position;

  num = 1.2;
  position.x_pos = 3.3;    // .의 의미 : position(구조체) 안의 x_pos 을 보겠다.
  position.y_pos = 7.7;

  printf("sizeof(position) = %lu\n", sizeof(position));
  printf("%lf\n", position.x_pos);
  printf("%lf\n", position.y_pos);

  return 0;
}
```

5. 함수포인터
함수의 주소를 저장할 수 있는 포인터(함수도 포인터에 저장가능)

Ex>
/*
함수포인터

void(*p)(void):
void 를 리턴하고 void 를 인자로 취하는 함수의 주소값을 저장할 수 있는 변수 : p

*/

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

```
void aaa(void)
{
  printf("aaa called\n");
}

void bbb(void(*p)(void))
{
  p();
  printf("bbb called\n");
}

int main(void)
{
  bbb(aaa);
  return 0;
}
```

6. 프로토타입
리턴, 함수명, 인자에 대한 기술서

2018. 3. 2 금 – 7 회차
1. memmove()
메모리복사
memory move 에 합성어
메모리의 값을 복사할 때 사용
Ex>
//memmove 예제
// <strinf.h> 헤더파일기입

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

int main(void)
{
  int i = 0;
  int src[5] = {1, 2, 3, 4, 5};
  int dst[5];

  memmove(dst, src, sizeof(src));

  for(i = 0; i < 5; i++)
    printf("dst[%d] = %d\n", i, dst[i]);

return 0;

}
```

2. momcopy
memmove 보다 속도가 빠르나 메모리보호가 안됨
Ex>
//memcopy 예제

//memcopy 가 속도가 빠르므로 이것을 쓰다가 결과가 이상할경우(메모리보호가 안됨),
// 메모리가 보호되는 memmove 사용

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

int main(void)
{
  char src[30] = "This is amazing";
  char*dst = src + 3;

  printf("before memmove = %s\n", src);
  memcpy(dst, src, 3);
  printf("after memmove = %s\n", dst);

  return 0;
}
```

3. puts(), putchar()
4. get()
5. strlen()

문자열의 길이를 구하는데 사용

Ex>

6. strcpy()x – strncpy()

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

int main(int argc, char **argv)
{
   char src[20] = "abcdef";
   char dst[20];
   strncpy(dst, src, 3);   //strcpy 는 숫자지정을 안함 (dst, src)이렇게 작성 - 그러므로 남은 공간에 해킹가능성
이 존재 , strncpy 는 숫자를 지정
   printf("dst = %s\n", dst);
   return 0;
}
```

7. strcmpx – strncmp

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

int main(int argc, char **argv)
{
    char src[20] = "made in korea";
    char dst[20] = "made in china";
    if(!strncmp(src, dst, 8))              //strncmp 는 서로 같으면 0 이 나옴, 그러므로 else 를 사용하거나, !
(NOT)을 사용
        printf("src, dst 는 서로 같음\n");
    else
        printf("src, dst 는 서로 다름\n");
    return 0;
```

```
}
```

함수포인터관련 문제

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

#define EMPTY 0

struct node
{
  int data;
  struct node *link;
};
typedef struct node Stack;

Stack *get_node()
{
  Stack *tmp;
  tmp = (Stack *)malloc(sizeof(Stack));
  tmp -> link = EMPTY;
  return tmp;
}

void push(Stack **top, int data)
{
  Stack *tmp;
  tmp = *top;
  *top = get_node();
  (*top) -> data = data;
  (*top) -> link = tmp;
}

int pop(Stack **top)
{
  Stack *tmp;
  int num;
  tmp = *top;
  if(*top == EMPTY)
  {
      printf("Stack is empty!!!\n");
      return 0;
  }
  num = tmp -> data;
  *top = (*top) -> link;
  free(tmp);
  return num;
}

int main(void)
{
  Stack *top = EMPTY;
  push(&top, 10);
```

```
    push(&top, 20);
    push(&top, 30);
    printf("%d\n", pop(&top));
    printf("%d\n", pop(&top));
    printf("%d\n", pop(&top));
    printf("%d\n", pop(&top));
    return 0;
}
```