01_ft_print

1. ft_printf

- ft_printf의 시작.
- printf의 반환값은 문자열의 길이.
- 가변인자
  - ap : argument point, 현재 인자를 저장하고 있는 포인터의 위치
  - va_list : 가변인자의 목록, 가변인자의 시작주소를 가르키는 포인터를 선언.
  - va_start(ap, value) : va_start는 va_list를 초기화하는 역할, 두 번째 인자인 고정인수는 첫 번째 가변인자 주소를 알기위해서 필요.
  - va_arg(ap, type) : 가변 인자 포인터에서 특정 자료형 크기만큼 값을 가져온 후 다음 가변인자로 포인터 증가.
  - va_end(ap) : 가변 인자 처리가 끝났을 때 포인터를 NULL로 초기화.

```
int       ft_printf(const char *format, ...)
{
  va_list   ap;
  int     cnt;

  va_start(ap, format);
  cnt = ft_start_printf(format, ap);
  va_end(ap);
  return (cnt);
}
```

1.1 ft_print_start

- format의 형태
  - "%[flag][width][.precision][type]"
  - flag/width/precision/type을 저장하기 위한 구조체 선언 및 초기화.
- '%'아닌 경우 일반 문자와 같이 출력.
- '%'일 경우 각 옵션을 확인하고 구조체의 값을 변경한다.
  - type을 만나기 전에는 flag/width/precision과 같은 옵션이므로 이를 확인하다.
  - parsing이 끝난 후 '0' flag 초기화.
    - 왼쪽정렬과 precision과 같이 사용할 수 없다.
      - 왼쪽정렬의 우선순위가 더 높다.
        - 왼쪽정렬일 때 0을 출력하면 출력값을 제대로 알지 못할 경우가 발생. (기대값 : 123, 출력값 : 12300)
        - precision이 들어온 경우 precision의 크기만큼 0을 채우기 때문에 같이 사용할 수 없다.(기대값 : 00123, 출력값 : 000000123)
    - 같이 사용되는 한 가지 경우는 % type.
  - parsing과 초기화가 끝난 후 각 type에 맞는 출력을 해준다.

```
static int    ft_start_printf(const char *format, va_list ap)
{
  int     s_cnt;
  t_format  *op;
```

```
  s_cnt = 0;
  if (!(op = (t_format *)malloc(sizeof(t_format))))
    return (-1);
  while (*format)
  {
    if (*format == '%')
    {
      ++format;
      //flag/width/precision/type을 저장하기 위한 구조체 선언 및 초기화
      ft_init_list(op);
      while (*format && !ft_strchr("cspdiuxX%", *format))
        ft_parse_format(*(format++), ap, op);
      if ((op->left == 1 || op->prec > -1) && *format != '%')
        op->zero = 0;
      s_cnt += ft_parse_type(*(format++), ap, op);
    }
    else
      s_cnt += ft_putchar(*(format++));
  }
  free(op);
  return (s_cnt);
}
```

1.2 ft_prase_format & ft_parse_width_prec

- type을 만나기 전 flag/width/precision와 같은 옵션을 확인하는 함수.
- '0' flag
  - width가 나오기전 선언이 되어야 하며, precision이 들어오지 않아야 한다.
  - precision이 들어오거나 '-' flag가 활성화 되면 사용할 수 없다.
- '-' flag
  - 왼쪽정렬
- '.'
  - precision 활성화.
  - 구조체의 precision의 초기값을 -1로 설정한 이유
    - precision = 0은 precision이 없다는 뜻이 아니라 precision의 값이 0이라는 의미.
    - '.' 뒤에 아무것도 없는 경우 .0과 같은 역할.
- number & '*'
  - width와 precision의 값.
    - '*'을 이용해 직접 입력할 수 있다.
  - width < 0
    - 값을 양수로 바꾸고 '-' flag를 활성화한다.
  - precision < 0
    - precision의 값에 상관없이 precision이 작동하지 않는 것으로 간주한다.

```
static void   ft_parse_width_prec(char format, va_list ap, t_format *op)
{
  if (ft_isdigit(format))
  {
    if (op->prec == -1)
      op->width = (op->width * 10) + format - '0';
    else
      op->prec = (op->prec * 10) + format - '0';
  }
  else if (format == '*')
  {
```

```
    if (op->prec == -1)
    {
      if ((op->width = va_arg(ap, int)) < 0)
      {
        op->width *= -1;
        op->left = 1;
      }
    }
    else
    {
      if ((op->prec = va_arg(ap, int)) < 0)
        op->prec = -1;
    }
  }
}

static void   ft_parse_format(char format, va_list ap, t_format *op)
{
  if (format == '0' && op->prec == -1 && op->width == 0)
    op->zero = 1;
  else if (format == '-')
    op->left = 1;
  else if (format == '.')
    op->prec = 0;
  else if (ft_isdigit(format) || format == '*')
    ft_parse_width_prec(format, ap, op);
}
```

1.3 ft_prase_type

- 각 type에 관한 출력을 위한 함수를 호출하는 함수.

```
static int    ft_parse_type(char format, va_list ap, t_format *op)
{
  op->type = format;
  if (format == 'c')
    return (ft_print_char(ap, op));
  if (format == 's')
    return (ft_print_str(ap, op));
  if (format == 'p')
    return (ft_print_ptr(ap, op));
  if (format == 'd' || format == 'i')
    return (ft_print_int(ap, op));
  if (format == 'u')
    return (ft_print_un_int(ap, op));
  if (format == 'x' || format == 'X')
    return (ft_print_hex(ap, op));
  if (format == '%')
    return (ft_print_per(op));
  return (0);
}
```

2. ft_print_char

- 'c' type을 출력하는 함수.
- va_arg type은 int.(가변인자로 받은 값이 int보다 작으면 int로, float은 double로 지정)
- 'c' type은 '0' flag과 precision의 영향을 받지 않는다.
- op→width -1을 해준 이유 : 배열의 index를 생각.

```
int ft_print_char(va_list ap, t_format *op)
{
  int cnt;
  int s_width;

  cnt = 0;
```

```
  s_width = 0;
  if (op->left)
  {
    cnt += ft_putchar(va_arg(ap, int));
    while (s_width++ < (op->width - 1))
      cnt += ft_putchar(' ');
  }
  else
  {
    while (s_width++ < (op->width - 1))
      cnt += ft_putchar(' ');
    cnt += ft_putchar(va_arg(ap, int));
  }
  return (cnt);
}
```

3. ft_print_str

- 's' type을 출력하는 함수.
- va_arg type은 char *.
- 문자열이 비어있는 경우 "(null)"을 반환.
- precision이 활성화되고 len가 precision보다 긴 경우
  - 문자열을 precision만큼 잘라서 출력한다.
  - len가 더 작은 경우, 문자열 그래도 출력한다.
- precision을 적용하여 자른 문자열의 길이와 width를 비교해 그 차이만큼 공백을 채워준다.
  - 문자열의 길이가 width가 더 길면 전체의 출력하는 폭이 문자열의 길이와 같으므로 op→width에 문자열의 길이를 저장한다.

```
int     ft_print_str(va_list ap, t_format *op)
{
  int   cnt;
  int   len;
  char  *tmp;

  tmp = va_arg(ap, char *);
  if (!tmp)
    tmp = "(null)";
  len = ft_strlen(tmp);
  if (len > op->prec && op->prec > -1)
    len = op->prec;
  if (len > op->width)
    op->width = len;
  cnt = ft_print_res(tmp, len, op);
  return (cnt);
}
```

3.1 ft_print_res

- 모든 type의 출력에서 사용되는 함수.
- 각 type 출력에서 문자열과 width를 비교
  - 문자열이 더 길면 전체 출력 폭이 문자열의 길이와 같으므로 op→width에 문자열의 길이가 저장.
    - op→width와 len가 같은 값을 가짐. 즉, 문자열의 크기와 폭의 크기가 같기때문에 시작하는 위치가 동일.
  - width가 더 길면 폭에서 문자열의 뺀 길이의 index부터 문자열의 길이만큼 채워넣음.(width = 10, len = 7 : "000abcdefg", index = 3부터 문자열을 채워넣음)
- 문자열 tmp는 precision을 적용한 상태.

    - 숫자는 precision만큼 0을 채워넣음.
    - 문자열은 precision만큼 문자열을 자름.
- '0' flag
    - 왼쪽정렬이 비활성 상태이며 precision이 없는 경우 사용.
- '-' flag
    - 비활성화 : 공백으로 문자열을 초기화하고, width와 문자열 길이 차이만큼의 index에서 시작.
    - 활성화 : 왼쪽부터 채워넣으면 되므로 index = 0부터 문자열 복사.

```
static int  ft_print_res(char *tmp, int len, t_format *op)
{
  int   cnt;
  char  *res;

  if (!(res = (char *)malloc(sizeof(char) * (op->width + 1))))
    return (0);
  res[op->width] = '\0';
  if (op->prec < 0 && op->zero && !op->left)
  {
    ft_memset(res, '0', op->width);
    ft_memcpy(res + op->width - len, tmp, len);
  }
  else if (!op->left)
  {
    ft_memset(res, ' ', op->width);
    ft_memcpy(res + op->width - len, tmp, len);
  }
  else
  {
    ft_memset(res, ' ', op->width);
    ft_memcpy(res, tmp, len);
  }
  cnt = ft_putstr(res);
  free(res);
  return (cnt);
}
```

4. ft_print_int

- 'd' & 'i' type을 출력하는 함수.
    - 부호가 있는 10진수 정수.
- va_arg type은 int.
- 모든 숫자 타입 출력 순서는 동일
    - itoa를 이용하여 입력받은 숫자를 문자열로 변환.
    - precision을 우선적으로 적용하여 precision과 문자열 길이 차이만큼 0을 출력.
        - precision뿐만 아니라 '0' flag인 경우 0을 출력하기 위해 같이 처리함.
    - 0을 적용하여 만든 문자열과 width를 비교하여 전체 폭을 결정하고 차이만큼 공백을 출력.
    - 출력한 문자열(글자 수) 반환.
    - '-' flag / '0' flag / 음수값 / 특수한 예외(NULL, 0값 처리) 등의 처리를 추가.
- 0을 채워넣어야 하는 조건에 맞춰 문자열의 길이를 구한다.
- 구한 문자열의 길이만큼 0을 채운 새로운 문자열을 만든다.
- 0을 채운 문자열의 길이와 width를 비교해 전체 문자열의 폭을 결정한다.

```
int     ft_print_int(va_list ap, t_format *op)
{
  char  *n_str;
  char  *tmp;
  int   len;
  int   cnt;

  n_str = ft_itoa(va_arg(ap, int));
  len = ft_calc_width(n_str, op);
  tmp = ft_apply_zero(n_str, len, op);
  len = ft_strlen(tmp);
  if (len > op->width)
    op->width = len;
  cnt = ft_print_res(tmp, len, op);
  free(n_str);
  free(tmp);
  return (cnt);
}
```

4.1 ft_calc_width

- 문자열의 길이를 구하고 '-' 부호가 들어왔을 경우 구조체의 sign값을 변경해준다.
- precision외에 0을 채워 넣어야하는 '0' flag인 경우
  - width가 문자열의 길이보다 커야 0을 채워넣을 수 있다.
  - 부호가 있을 경우 이를 적용해주지 않으면 '-' 부호 앞에 0이 먼저 오는 경우가 발생한다.(str = "-123", len = 4, width = 8 : output = "0000-123")
  - 부호가 없을 경우에는 이 조건이 없어도 된다.
- precision의 값이 0이고 들어온 인자가 0이면 빈 문자열을 반환해야한다.
  - 이 때, 인자로 들어온 값이 0이어야 하므로 문자열의 길이가 1인 조건을 추가.
- precision이 들어온 경우
  - 부호를 뺀 문자열의 길이보다 precision이 크면 precision의 크기에 부호를 더한 값이 길이가 된다.("%.5d", -123 : -00123)

```
static int  ft_calc_width(char *n_str, t_format *op)
{
  int   len;

  len = ft_strlen(n_str);
  if (*n_str == '-')
    op->sign = 1;
  if (op->prec < 0 && op->zero && op->width > len)
    len = op->width;
  if (op->prec == 0 && *n_str == '0' && len == 1)
    len = 0;
  if (op->prec > len - op->sign)
    len = op->prec + op->sign;
  return (len);
}
```

4.2 ft_apply_zero

- 0을 채워야하는 길이만큼 0을 채운 새로운 문자열을 만드는 함수.
- 문자열의 전체길이와 부호를 뺀 문자의 길이(숫자만 있는 길이)를 각각 구한다.
  - 부호가 없는 경우 사용하지않지만 모든 숫자 타입에 같은 코드를 적용.
  - ft_calc_width 함수에서 구한 길이로 새로운 문자열을 만듬.

- 부호가 존재하면 문자열의 0번째 index에 부호를 추가.(di type에만 적용)

```
static char *ft_apply_zero(char *n_str, int len, t_format *op)
{
  int   s_len;
  int   n_len;
  char  *res;

  s_len = ft_strlen(n_str);
  n_len = s_len - op->sign;
  if (len == 0)
    return (ft_strdup(""));
  if (!(res = (char *)malloc(sizeof(char) * (len + 1))))
    return (NULL);
  ft_memset(res, '0', len);
  ft_memcpy(res + len - n_len, n_str + op->sign, n_len);
  if (op->sign)
    res[0] = '-';
  res[len] = '\0';
  return (res);
}
```

4.3 ft_print_res

- 3.1 에서 정리한 내용과 동일.

```
static int  ft_print_res(char *tmp, int len, t_format *op)
{
  char  *res;
  int   cnt;

  if (!(res = (char *)malloc(sizeof(char) * (op->width + 1))))
    return (-1);
  if (op->prec < 0 && op->zero && !op->left)
  {
    ft_memset(res, '0', op->width);
    ft_memcpy(res + op->width - len, tmp, len);
  }
  else if (!op->left)
  {
    ft_memset(res, ' ', op->width);
    ft_memcpy(res + op->width - len, tmp, len);
  }
  else
  {
    ft_memset(res, ' ', op->width);
    ft_memcpy(res, tmp, len);
  }
  res[op->width] = '\0';
  cnt = ft_putstr(res);
  free(res);
  return (cnt);
}
```

5. ft_print_un_int

- 'u' type을 출력하는 함수.
  - 부호가 없는 10진수 정수.
- va_arg type은 unsigned int.
- itoa대신 lltoa함수 사용한다.
  - itoa는 int형으로 받으므로 long long type으로 받는 lltoa를 사용.

```
int     ft_print_un_int(va_list ap, t_format *op)
{
  char  *n_str;
  char  *tmp;
  int   len;
  int   cnt;

  n_str = ft_lltoa(va_arg(ap, unsigned int));
  len = ft_calc_width(n_str, op);
  tmp = ft_apply_zero(n_str, len, op);
  len = ft_strlen(tmp);
  if (len > op->width)
    op->width = len;
  cnt = ft_print_res(tmp, len, op);
  free(n_str);
  free(tmp);
  return (cnt);
}
```

6. ft_print_hex

- 'x' & 'X' type을 출력하는 함수.
  - 부호가 없는 16진수 정수.
  - 'x' : 소문자, 'X' : 대문자
- va_arg type은 unsigned int.
- 16진수로 변환하여 출력한다.
  - 10진수로 들어온 입력값을 ft_convert_base 함수를 이용하여 16진수로 변환.

```
int     ft_print_hex(va_list ap, t_format *op)
{
  char  *n_str;
  char  *tmp;
  int   len;
  int   cnt;

  if (op->type == 'x')
    n_str = ft_convert_base(va_arg(ap, unsigned int), "0123456789abcdef");
  if (op->type == 'X')
    n_str = ft_convert_base(va_arg(ap, unsigned int), "0123456789ABCDEF");
  len = ft_calc_width(n_str, op);
  tmp = ft_apply_zero(n_str, len, op);
  len = ft_strlen(tmp);
  if (len > op->width)
    op->width = len;
  cnt = ft_print_res(tmp, len, op);
  free(n_str);
  free(tmp);
  return (cnt);
}
```

6.1 ft_convert_base

- 입력값을 진법으로 들어온 convert에 맞게 변환하는 함수.
- 16진수 : "0123456789abcdef"
  - 변환할 진법으로 들어온 문자의 길이를 구한다. 이는 진법의 길이(16진수 = 16)
  - 변환된 진법을 저장하기위한 문자열의 길이를 구한다.
    - 1234가 입력값으로 들어온 경우 : n = 1234, base = 16.

        - ft_base_len 함수에서 몫이 0이 될 때 까지 n/base를 반복하여 i를 증가시킨다.
        - 3번을 나눌 수 있으므로 i의 값은 3이 된다.
    - len의 크기만큼 변환된 진법을 저장할 문자열을 만든다.
    - 변환할 숫자를 진법으로 들어온 문자열의 길이로 나눈 나머지의 인덱스에 있는 문자가 뒤에서부터 입력된다.
        - 1234%16 = 2, 77%16 = 13 = d, 4%16 = 4 : output = "4d2"

```
static int  ft_base_len(unsigned long long n, int base)
{
  int i;

  i = 0;
  if (n <= 0)
    i++;
  while (n)
  {
    n /= base;
    i++;
  }
  return (i);
}

char    *ft_convert_base(unsigned long long n, char *convert)
{
  unsigned long long  nbr;
  char        *res;
  int         len;
  int         base;

  base = ft_strlen(convert);
  len = ft_base_len(n, base);
  nbr = n;
  if (!(res = (char *)malloc(sizeof(char) * (len + 1))))
    return (NULL);
  res[len] = '\0';
  while (--len > 0 && nbr != 0)
  {
    res[len] = convert[nbr % base];
    nbr /= base;
  }
  res[0] = convert[nbr % base];
  return (res);
}
```

7. ft_print_ptr

- 'p' type을 출력하는 함수.
- va_arg type은 void *. (to long long type casting)
- pointer 출력은 다른 타입과 다르게 "0x"를 추가해줘야 한다.
    - 0을 적용시킨 새로운 문자열에 "0x"를 추가해준 후 새롭게 문자열의 길이를 구한다.
    - ft_calc_width 함수에서 precision과 기존 문자열의 길이만 +2를 해주면 되는 줄 알았다.
        - 이 함수에서는 0을 적용시키는 문자의 길이를 구하는 것이며 0을 추가한 새로운 문자열을 만든 후 "0x"를 추가한 후 len + 2를 해줘야한다.
        - vnc의 기준에 맞추면 width일 때도 + 2를 해주는 것이 맞는 듯 하다.
- vnc gcc: : "0x" + 16진수 12자리 (소문자)
- ssh gcc : "0x" + 16진수 10자리 (소문자)
- win은 또 다르고, 컴파일 환경마다 다르다. 이번 과제는 vnc gcc 기준.
- tmp = ft_strjoin("0x", tmp)를 했을 경우 메모리가 free되지 않고 남아있어 n_str을 free 시킨 후 n_str에 저장해줬다.

- 다른 숫자 타입에서는 tmp를 넘기지만 pointer에서는 n_str을 넘김.

```
int     ft_print_ptr(va_list ap, t_format *op)
{
  unsigned long long  num;
  char        *n_str;
  char        *tmp;
  int         len;
  int         cnt;

  num = (long long)(va_arg(ap, void *));
  n_str = ft_convert_base(num, "0123456789abcdef");
  len = ft_calc_width(n_str, op);
  tmp = ft_apply_zero(n_str, len, op);
  free(n_str);
  n_str = ft_strjoin("0x", tmp);
  len = ft_strlen(n_str);
  if (len > op->width)
    op->width = len;
  cnt = ft_print_res(n_str, len, op);
  free(n_str);
  free(tmp);
  return (cnt);
}
```

8. ft_print_per

- '%' type을 출력하는 함수.
- 문자 %를 출력하기 위해 사용된다.
- '0' flag와 '-'가 동시에 사용될 수 있다.
  - '-' flag 활성화
    - '0' flag는 적용되지 않는다.
  - '-' flag 비활성화
    - 오른쪽 정렬로 출력을 하며 이 때, '0' flag가 사용될 수 있다.

```
int   ft_print_per(t_format *op)
{
  int cnt;
  int s_width;

  cnt = 0;
  s_width = 0;
  if (op->left)
  {
    cnt += ft_putchar('%');
    while (s_width++ < (op->width - 1))
      cnt += ft_putchar(' ');
  }
  else
  {
    while (s_width++ < (op->width - 1))
    {
      if (op->zero)
        cnt += ft_putchar('0');
      else
        cnt += ft_putchar(' ');
    }
    cnt += ft_putchar('%');
  }
  return (cnt);
}
```

- 참고사이트
  - https://modoocode.com/35
  - https://jhnyang.tistory.com/293
  - https://dojang.io/mod/page/view.php?id=736
  - https://stdbc.tistory.com/m/59
  - [https://velog.io/@hidaehyunlee/형식태그와-서식지정자-printf-함수의-옵션-알아보기](https://velog.io/@hidaehyunlee/%ED%98%95%EC%8B%9D%ED%83%9C%EA%B7%B8%EC%99%80-%EC%84%9C%EC%8B%9D%EC%A7%80%EC%A0%95%EC%9E%90-printf-%ED%95%A8%EC%88%98%EC%9D%98-%EC%98%B5%EC%85%98-%EC%95%8C%EC%95%84%EB%B3%B4%EA%B8%B0)

printf