2020. 07. 06

malloc()

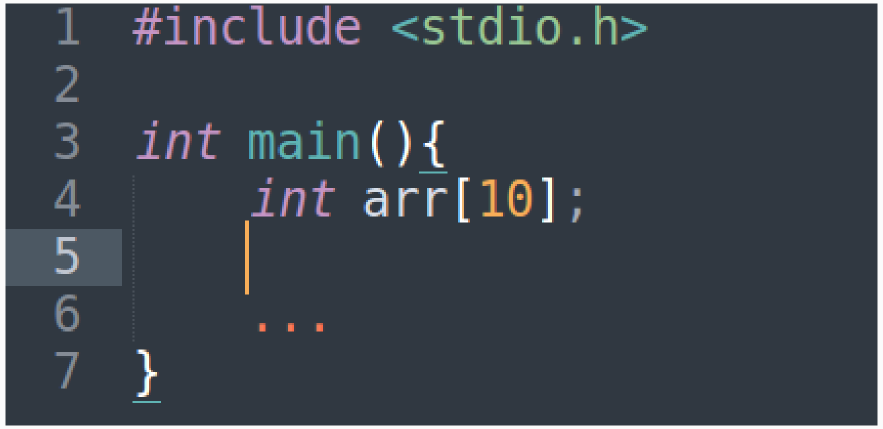
SCP 이예준

Contents

malloc(), 동적 메모리 할당

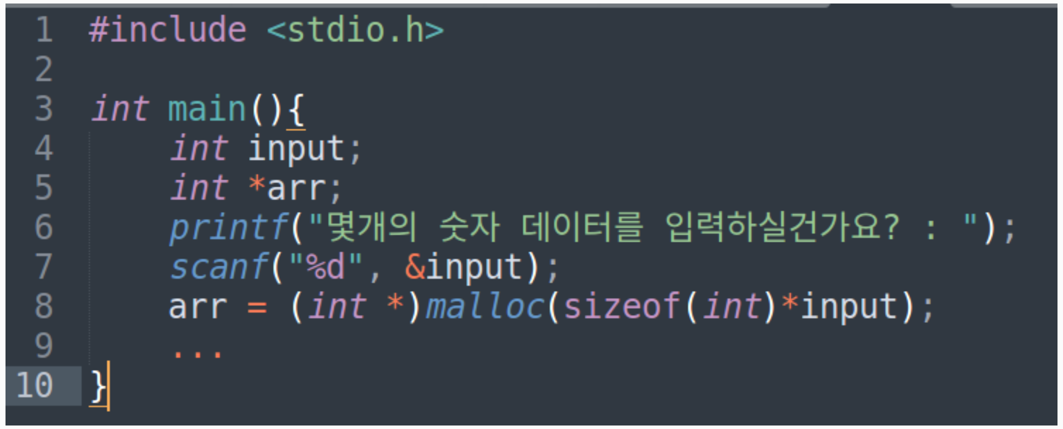
- 동적 메모리 할당을 하기 위해서 사용되는 함수.
- 사용자가 마음대로 할당하여 사용하고 해제할 수 있다.

정적 메모리 할당



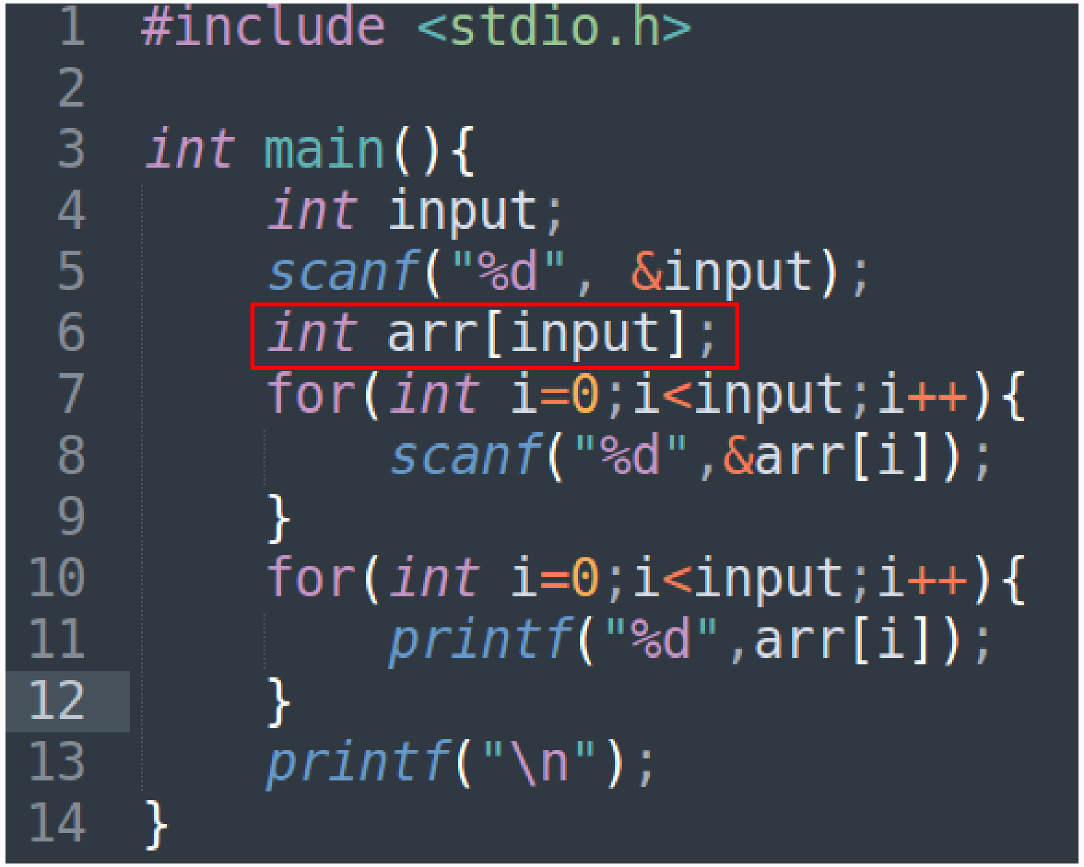
```c
#include <stdio.h>

int main(){
    int arr[10];

    ...
}
```

동적 메모리 할당



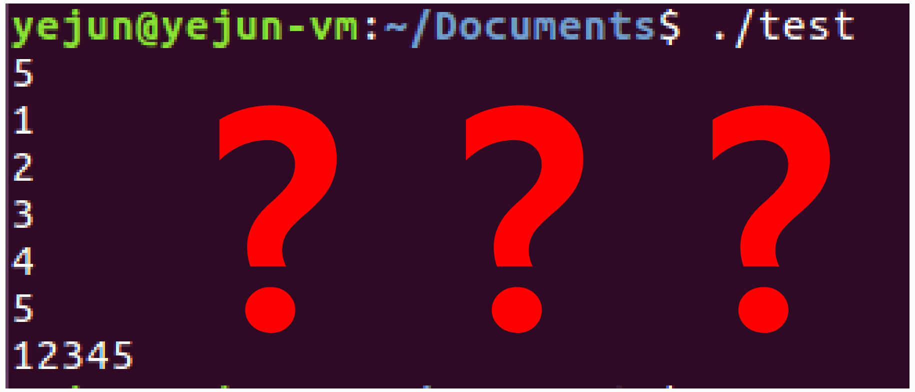
```c
#include <stdio.h>

int main(){
    int input;
    int *arr;
    printf("몇개의 숫자 데이터를 입력하실건가요? : ");
    scanf("%d", &input);
    arr = (int *)malloc(sizeof(int)*input);
    ...
}
```

문제



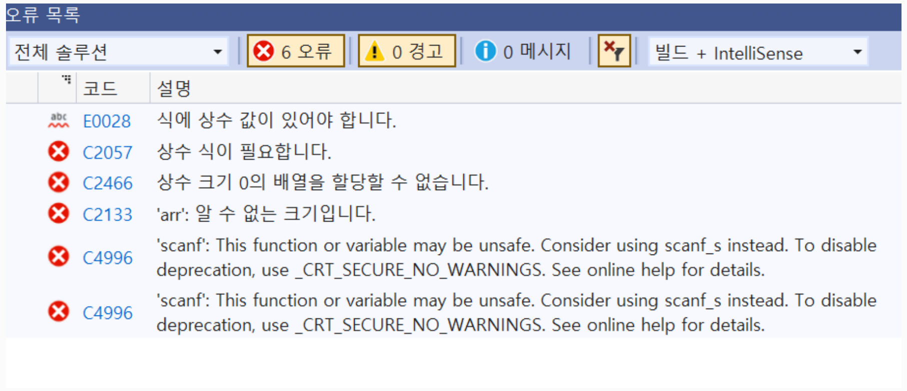
```c
#include <stdio.h>

int main(){
	int input;
	scanf("%d", &input);
	int arr[input];
	for(int i=0;i<input;i++){
		scanf("%d",&arr[i]);
	}
	for(int i=0;i<input;i++){
		printf("%d",arr[i]);
	}
	printf("\n");
}
```

GCC



```
yejun@yejun-vm:~/Documents$ ./test
5
1
2
3
4
5
12345
```

MSVC



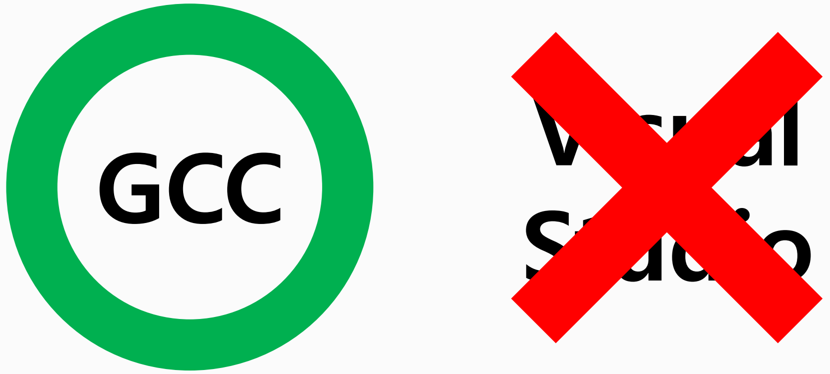
코드	설명
E0028	식에 상수 값이 있어야 합니다.
C2057	상수 식이 필요합니다.
C2466	상수 크기 0의 배열을 할당할 수 없습니다.
C2133	'arr': 알 수 없는 크기입니다.
C4996	'scanf': This function or variable may be unsafe. Consider using scanf_s instead. To disable deprecation, use _CRT_SECURE_NO_WARNINGS. See online help for details.
C4996	'scanf': This function or variable may be unsafe. Consider using scanf_s instead. To disable deprecation, use _CRT_SECURE_NO_WARNINGS. See online help for details.

가변 길이 배열(Variable-length Array)

배열의 크기를 컴파일 타임에 정하지 않고 실행 타임에 정할 수 있도록 하는 기능

C언어의 표준

- C99 : ANSI 표준화 이후, 1999년에 발표된 표준 -> VLA 지원
- C11 : 2011년 발표된 표준 -> VLA 지원이 필수가 아님(msvc은 지원X)
- C++ 에서 지원 안함.



가변 길이 배열(Variable-length Array)

malloc()
- heap 영역에 생성
- free()를 호출할 때 까지 프로그램에서 전역변수로 사용가능
- 스택보다 더 큰 메모리 할당이 가능함

VLA
- stack 영역에 생성
- 해당하는 스코프를 벗어나면 자동으로 메모리 관리 – 지역변수로 사용
- stack overflow 발생 위험

-> 개발자들은 'C99에 VLA 괜히 넣었다.' 라는 분위기이다.
(어찌됐건,, 그냥 malloc()을 배우면 된다.)

* 스코프 : 코드에 접근 할 수 있는 범위

메모리구조 – Heap

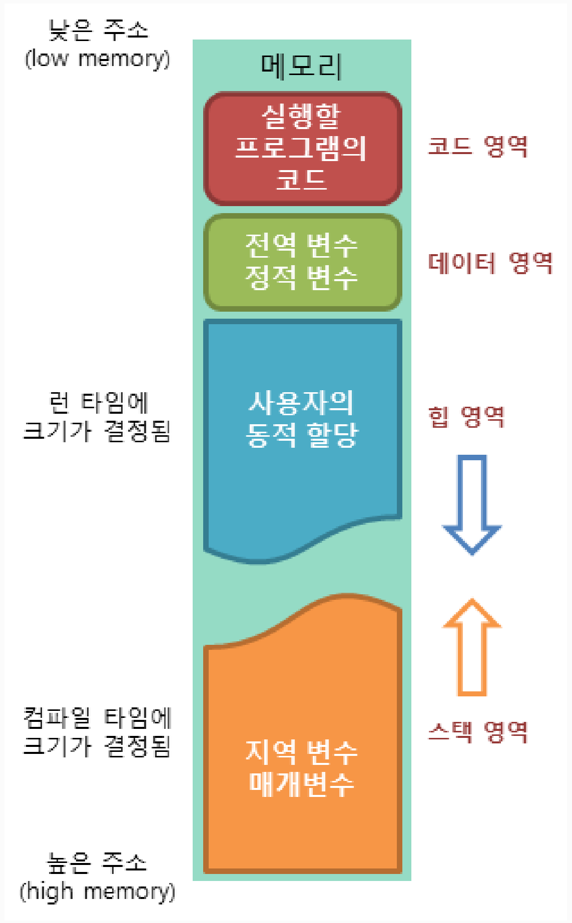
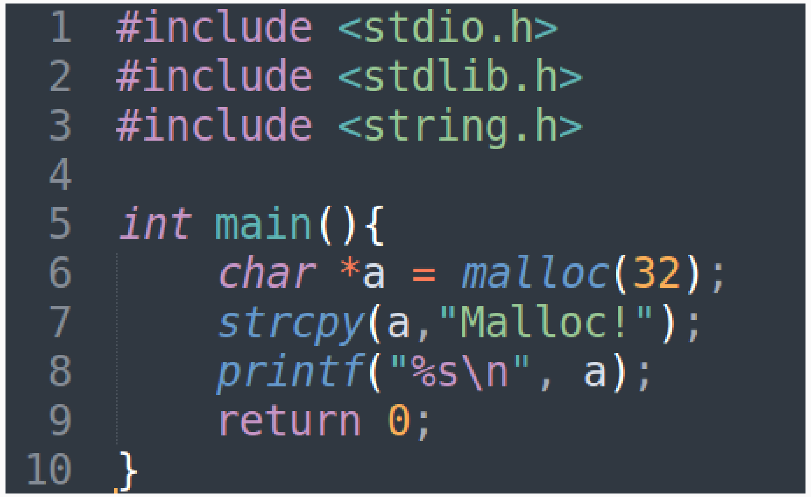
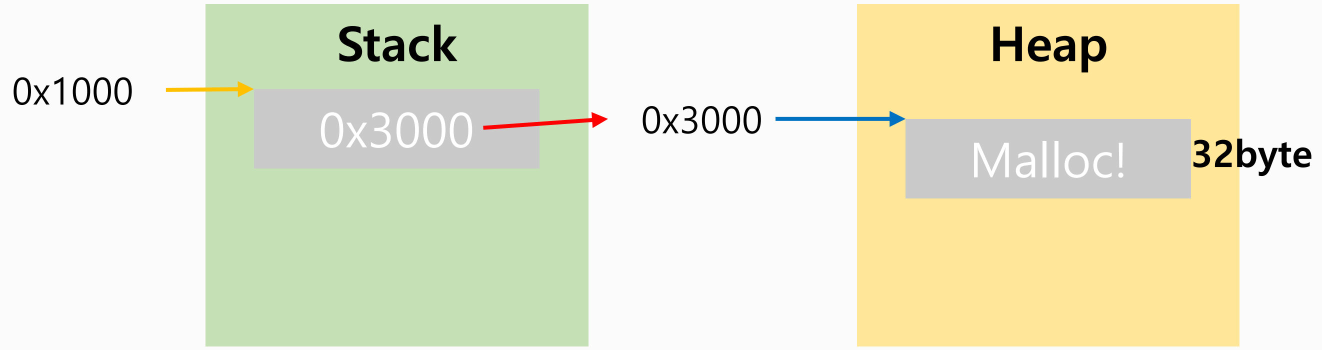


사진 출처 : https://tcpschool.com/c/c_memory_structure

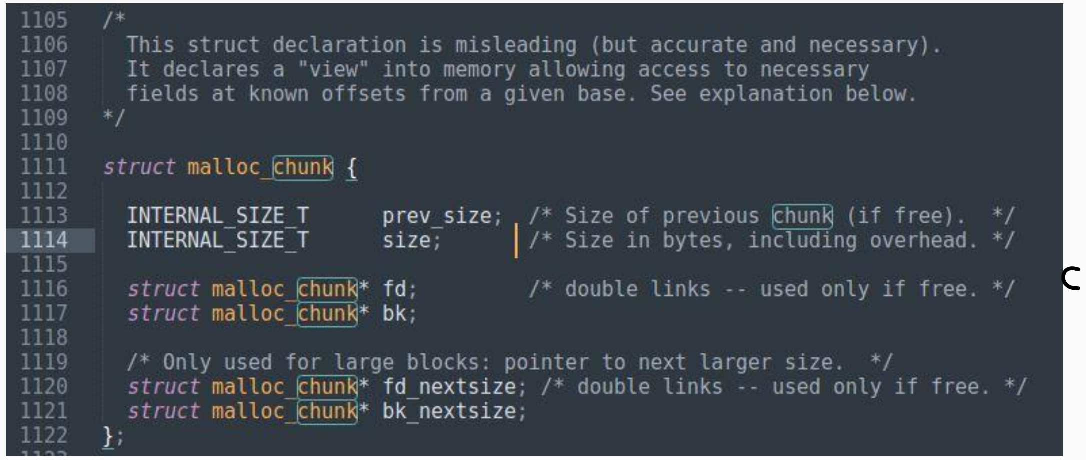


```c
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

int main(){
    char *a = malloc(32);
    strcpy(a,"Malloc!");
    printf("%s\n", a);
    return 0;
}
```



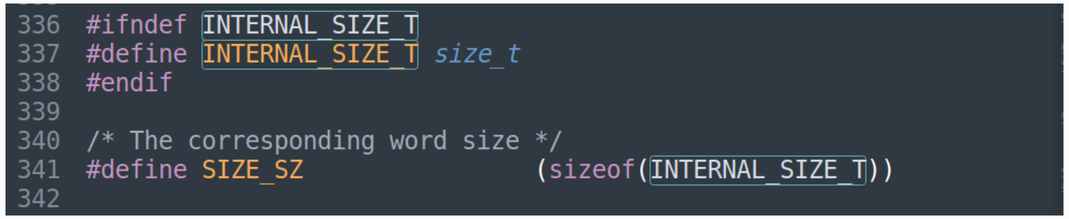
Malloc chunk의 구조

malloc.c : 1105

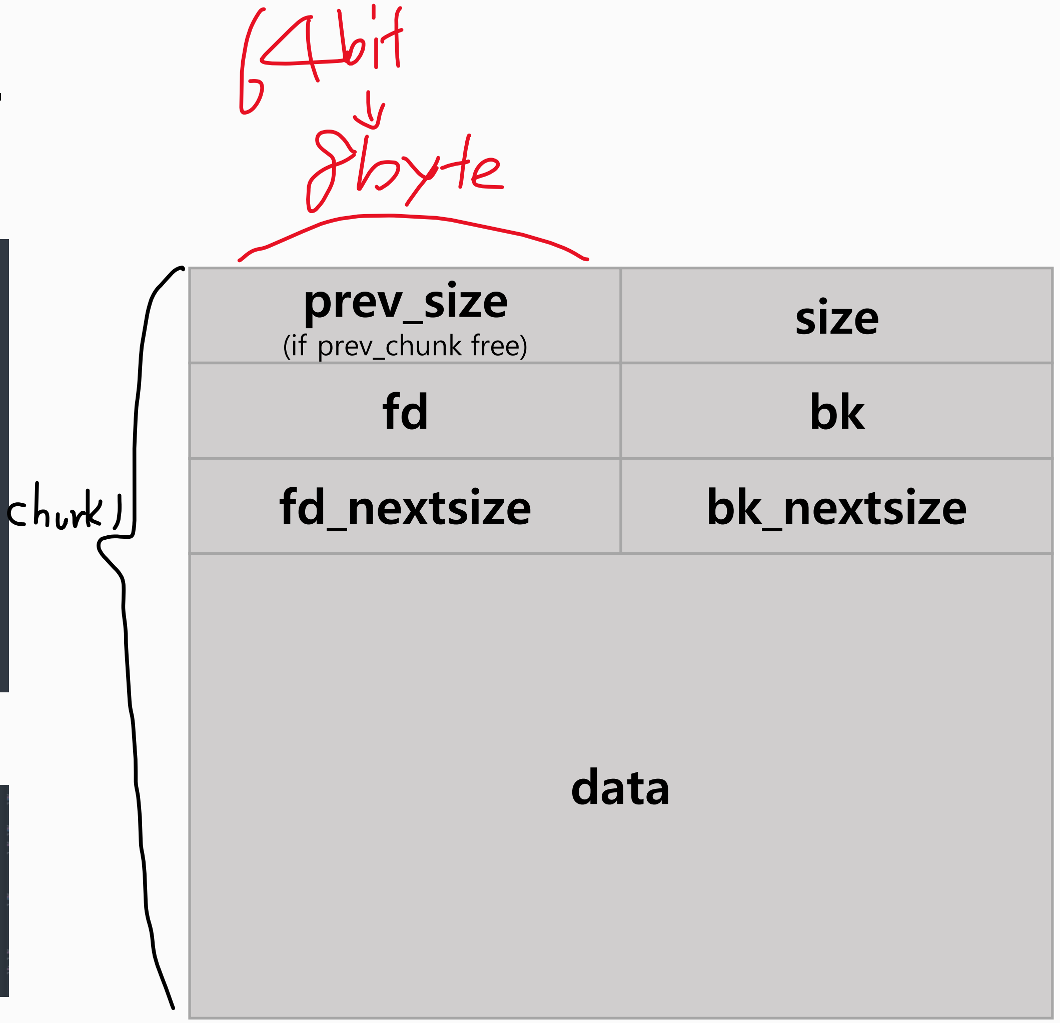


```
1105  /*
1106    This struct declaration is misleading (but accurate and necessary).
1107    It declares a "view" into memory allowing access to necessary
1108    fields at known offsets from a given base. See explanation below.
1109  */
1110
1111  struct malloc_chunk {
1112
1113    INTERNAL_SIZE_T      prev_size;  /* Size of previous chunk (if free).  */
1114    INTERNAL_SIZE_T      size;       /* Size in bytes, including overhead. */
1115
1116    struct malloc_chunk* fd;         /* double links -- used only if free. */
1117    struct malloc_chunk* bk;
1118
1119    /* Only used for large blocks: pointer to next larger size.  */
1120    struct malloc_chunk* fd_nextsize; /* double links -- used only if free. */
1121    struct malloc_chunk* bk_nextsize;
1122  };
```

malloc.c : 336

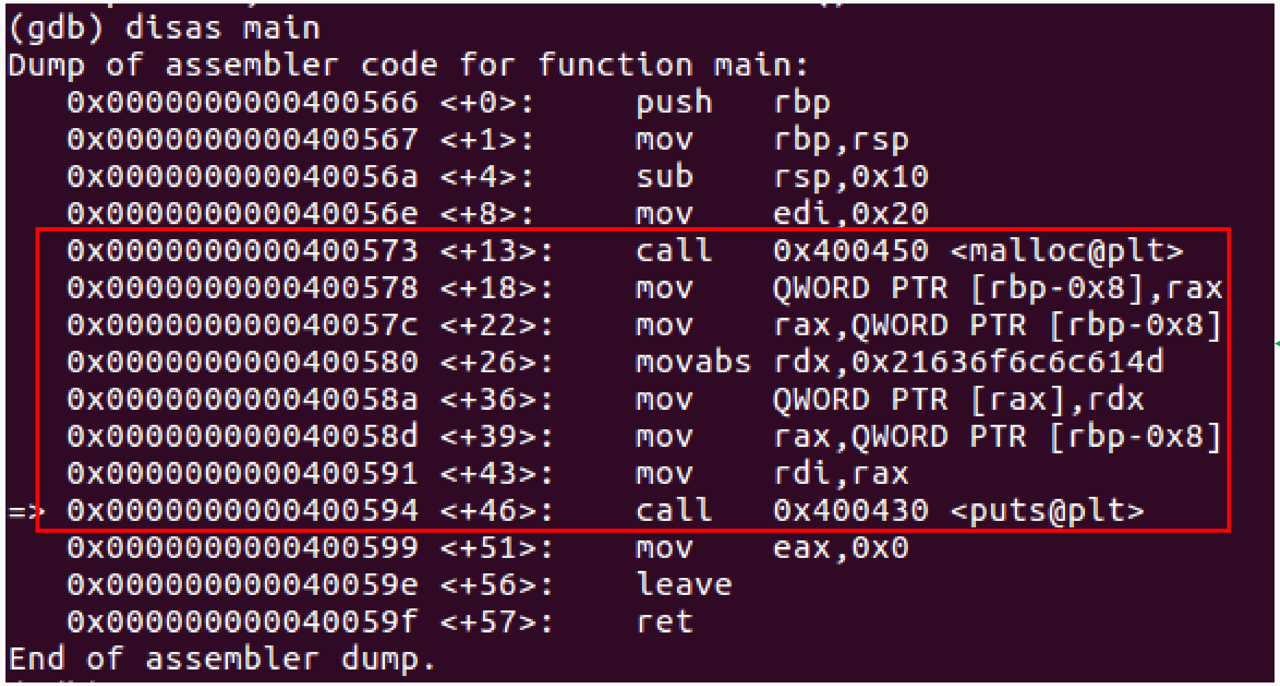


```
336  #ifndef INTERNAL_SIZE_T
337  #define INTERNAL_SIZE_T size_t
338  #endif
339
340  /* The corresponding word size */
341  #define SIZE_SZ                (sizeof(INTERNAL_SIZE_T))
342
```

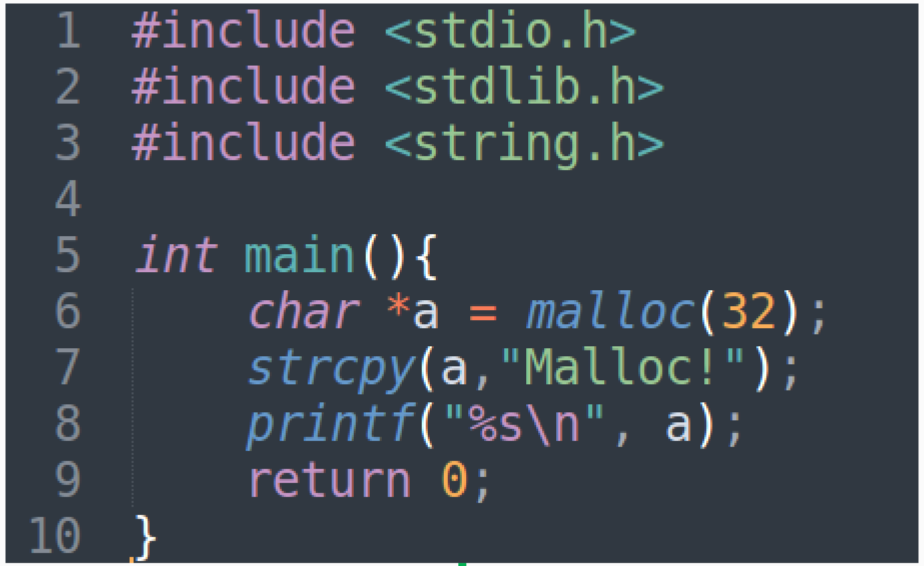


fd(forward pointer) : 할당이 해제된 다음 청크, bk(backward pointer) : 할당이 해제된 이전 청크.

메모리구조 – Heap

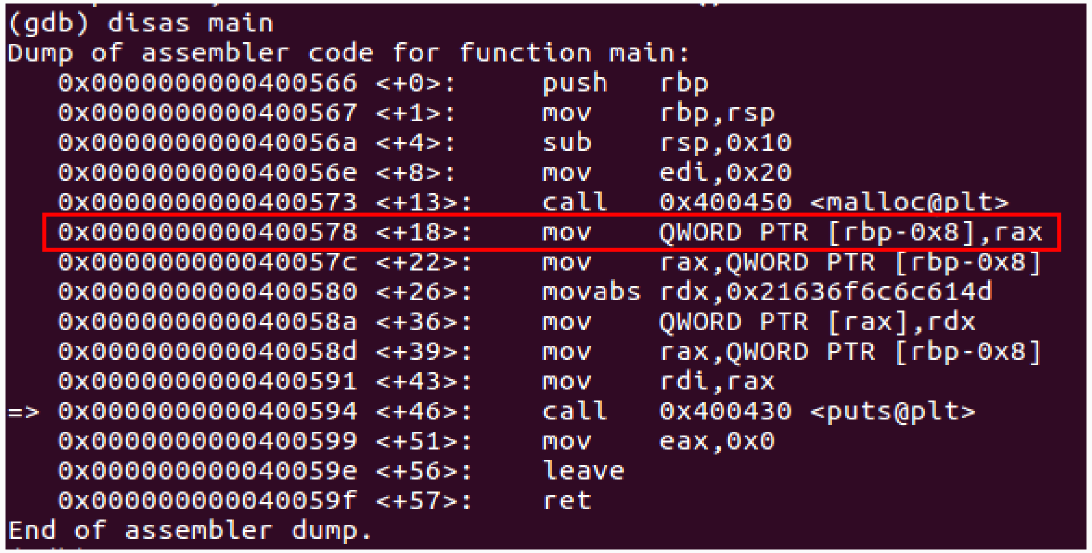


```
(gdb) disas main
Dump of assembler code for function main:
   0x0000000000400566 <+0>:     push   rbp
   0x0000000000400567 <+1>:     mov    rbp,rsp
   0x000000000040056a <+4>:     sub    rsp,0x10
   0x000000000040056e <+8>:     mov    edi,0x20
   0x0000000000400573 <+13>:    call   0x400450 <malloc@plt>
   0x0000000000400578 <+18>:    mov    QWORD PTR [rbp-0x8],rax
   0x000000000040057c <+22>:    mov    rax,QWORD PTR [rbp-0x8]
   0x0000000000400580 <+26>:    movabs rdx,0x21636f6c6c614d
   0x000000000040058a <+36>:    mov    QWORD PTR [rax],rdx
   0x000000000040058d <+39>:    mov    rax,QWORD PTR [rbp-0x8]
   0x0000000000400591 <+43>:    mov    rdi,rax
=> 0x0000000000400594 <+46>:    call   0x400430 <puts@plt>
   0x0000000000400599 <+51>:    mov    eax,0x0
   0x000000000040059e <+56>:    leave
   0x000000000040059f <+57>:    ret
End of assembler dump.
```

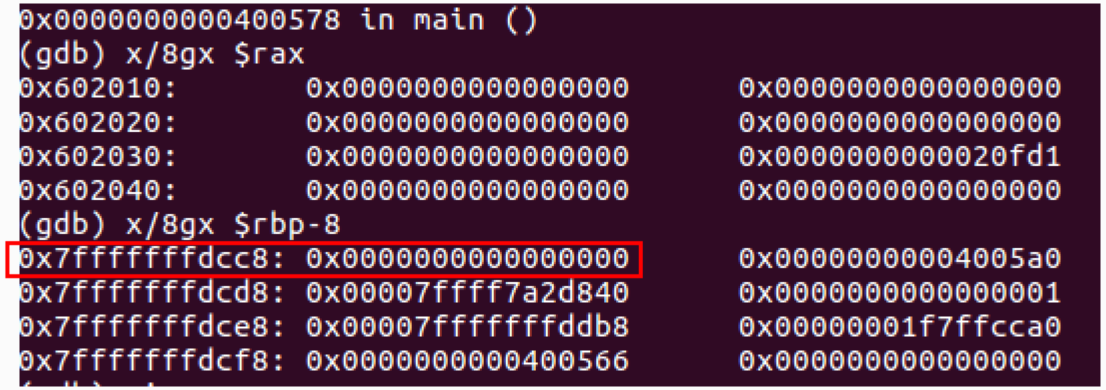


```c
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

int main(){
    char *a = malloc(32);
    strcpy(a,"Malloc!");
    printf("%s\n", a);
    return 0;
}
```

메모리구조 – Heap

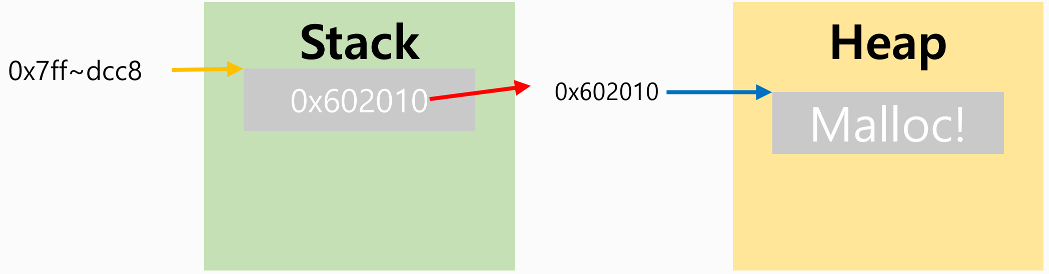


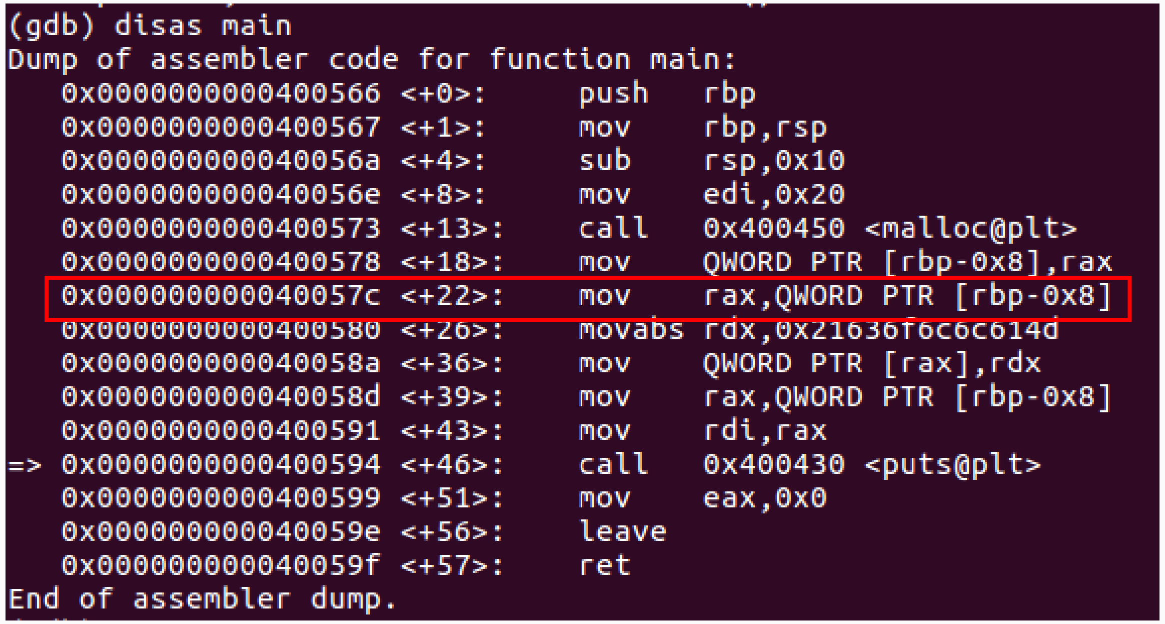
```
(gdb) disas main
Dump of assembler code for function main:
   0x0000000000400566 <+0>:     push   rbp
   0x0000000000400567 <+1>:     mov    rbp,rsp
   0x000000000040056a <+4>:     sub    rsp,0x10
   0x000000000040056e <+8>:     mov    edi,0x20
   0x0000000000400573 <+13>:    call   0x400450 <malloc@plt>
   0x0000000000400578 <+18>:    mov    QWORD PTR [rbp-0x8],rax
   0x000000000040057c <+22>:    mov    rax,QWORD PTR [rbp-0x8]
   0x0000000000400580 <+26>:    movabs rdx,0x21636f6c6c614d
   0x000000000040058a <+36>:    mov    QWORD PTR [rax],rdx
   0x000000000040058d <+39>:    mov    rax,QWORD PTR [rbp-0x8]
   0x0000000000400591 <+43>:    mov    rdi,rax
=> 0x0000000000400594 <+46>:    call   0x400430 <puts@plt>
   0x0000000000400599 <+51>:    mov    eax,0x0
   0x000000000040059e <+56>:    leave
   0x000000000040059f <+57>:    ret
End of assembler dump.
```



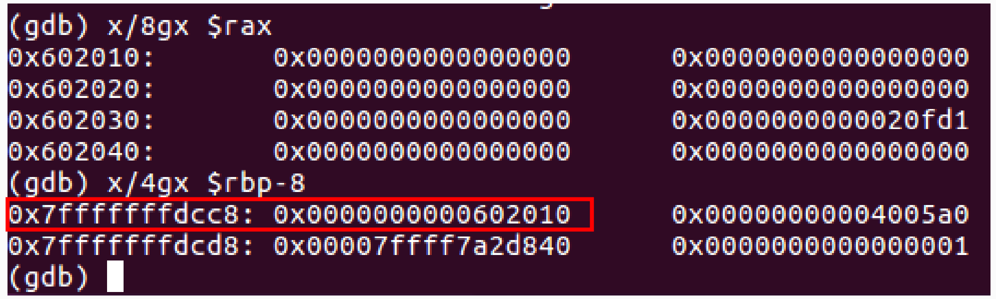
```
0x0000000000400578 in main ()
(gdb) x/8gx $rax
0x602010:       0x0000000000000000      0x0000000000000000
0x602020:       0x0000000000000000      0x0000000000000000
0x602030:       0x0000000000000000      0x0000000000020fd1
0x602040:       0x0000000000000000      0x0000000000000000
(gdb) x/8gx $rbp-8
0x7fffffffdcc8: 0x0000000000000000      0x00000000004005a0
0x7fffffffdcd8: 0x00007ffff7a2d840      0x0000000000000001
0x7fffffffdce8: 0x00007fffffffddb8      0x00000001f7ffcca0
0x7fffffffdcf8: 0x0000000000400566      0x0000000000000000
```

메모리구조 – Heap

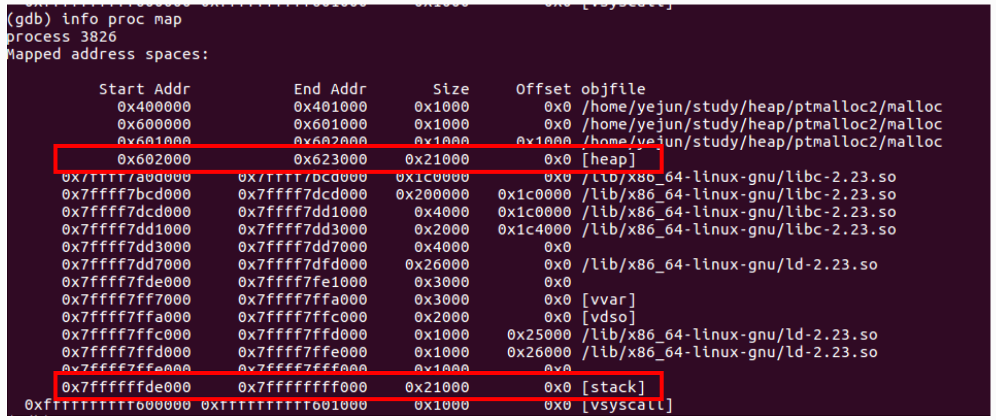




```
(gdb) disas main
Dump of assembler code for function main:
   0x0000000000400566 <+0>:     push   rbp
   0x0000000000400567 <+1>:     mov    rbp,rsp
   0x000000000040056a <+4>:     sub    rsp,0x10
   0x000000000040056e <+8>:     mov    edi,0x20
   0x0000000000400573 <+13>:    call   0x400450 <malloc@plt>
   0x0000000000400578 <+18>:    mov    QWORD PTR [rbp-0x8],rax
   0x000000000040057c <+22>:    mov    rax,QWORD PTR [rbp-0x8]
   0x0000000000400580 <+26>:    movabs rdx,0x21636f6c6c614d
   0x000000000040058a <+36>:    mov    QWORD PTR [rax],rdx
   0x000000000040058d <+39>:    mov    rax,QWORD PTR [rbp-0x8]
   0x0000000000400591 <+43>:    mov    rdi,rax
=> 0x0000000000400594 <+46>:    call   0x400430 <puts@plt>
   0x0000000000400599 <+51>:    mov    eax,0x0
   0x000000000040059e <+56>:    leave
   0x000000000040059f <+57>:    ret
End of assembler dump.
```

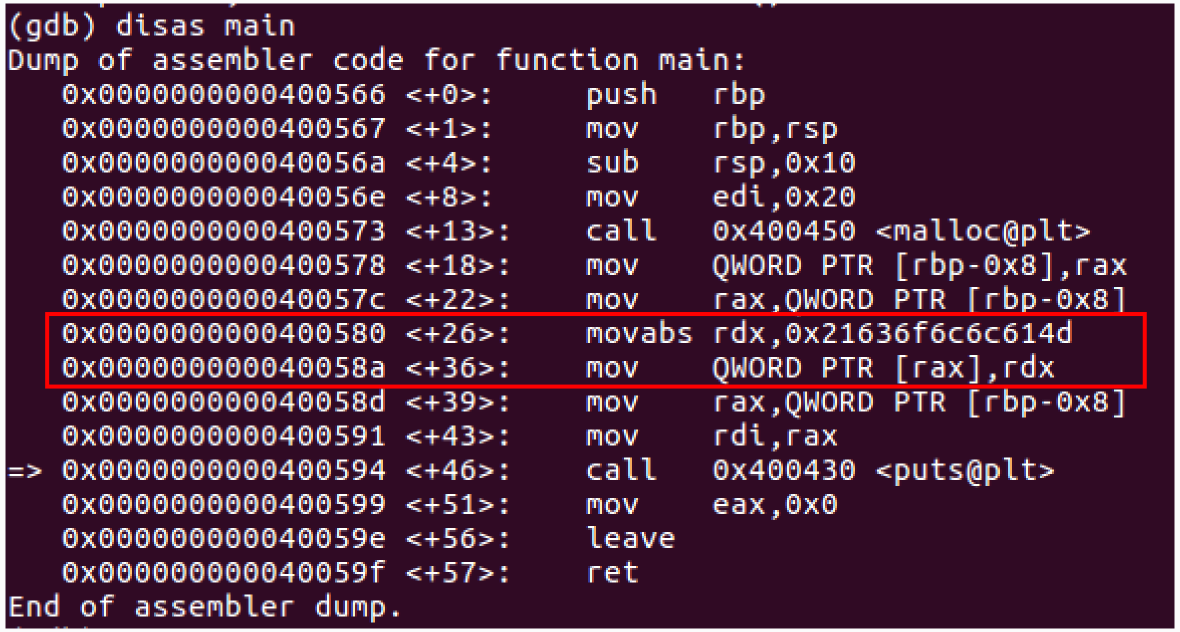


```
(gdb) x/8gx $rax
0x602010:       0x0000000000000000      0x0000000000000000
0x602020:       0x0000000000000000      0x0000000000000000
0x602030:       0x0000000000000000      0x0000000000020fd1
0x602040:       0x0000000000000000      0x0000000000000000
(gdb) x/4gx $rbp-8
0x7fffffffdcc8: 0x0000000000602010      0x00000000004005a0
0x7fffffffdcd8: 0x00007ffff7a2d840      0x0000000000000001
(gdb)
```

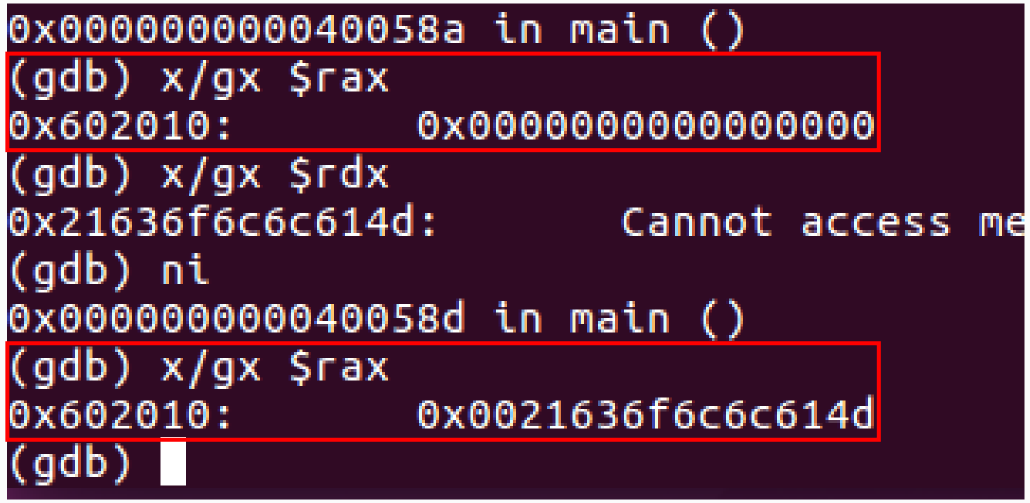


```
(gdb) info proc map
process 3826
Mapped address spaces:

          Start Addr           End Addr       Size     Offset objfile
            0x400000           0x401000     0x1000        0x0 /home/yejun/study/heap/ptmalloc2/malloc
            0x600000           0x601000     0x1000        0x0 /home/yejun/study/heap/ptmalloc2/malloc
            0x601000           0x602000     0x1000     0x1000 /home/yejun/study/heap/ptmalloc2/malloc
            0x602000           0x623000    0x21000        0x0 [heap]
      0x7ffff7a0d000     0x7ffff7bcd000   0x1c0000        0x0 /lib/x86_64-linux-gnu/libc-2.23.so
      0x7ffff7bcd000     0x7ffff7dcd000   0x200000   0x1c0000 /lib/x86_64-linux-gnu/libc-2.23.so
      0x7ffff7dcd000     0x7ffff7dd1000     0x4000   0x1c0000 /lib/x86_64-linux-gnu/libc-2.23.so
      0x7ffff7dd1000     0x7ffff7dd3000     0x2000   0x1c4000 /lib/x86_64-linux-gnu/libc-2.23.so
      0x7ffff7dd3000     0x7ffff7dd7000     0x4000        0x0
      0x7ffff7dd7000     0x7ffff7dfd000    0x26000        0x0 /lib/x86_64-linux-gnu/ld-2.23.so
      0x7ffff7fde000     0x7ffff7fe1000     0x3000        0x0
      0x7ffff7ff7000     0x7ffff7ffa000     0x3000        0x0 [vvar]
      0x7ffff7ffa000     0x7ffff7ffc000     0x2000        0x0 [vdso]
      0x7ffff7ffc000     0x7ffff7ffd000     0x1000    0x25000 /lib/x86_64-linux-gnu/ld-2.23.so
      0x7ffff7ffd000     0x7ffff7ffe000     0x1000    0x26000 /lib/x86_64-linux-gnu/ld-2.23.so
      0x7ffff7ffe000     0x7ffff7fff000     0x1000        0x0
      0x7ffffffde000     0x7ffffffff000    0x21000        0x0 [stack]
  0xffffffffff600000 0xffffffffff601000     0x1000        0x0 [vsyscall]
```

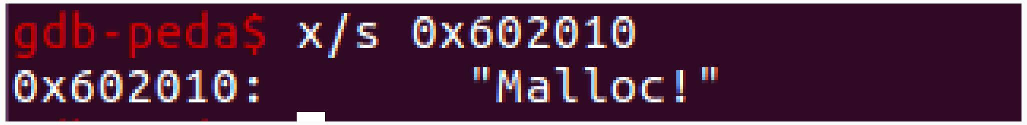
메모리구조 – Heap



```
(gdb) disas main
Dump of assembler code for function main:
   0x0000000000400566 <+0>:     push   rbp
   0x0000000000400567 <+1>:     mov    rbp,rsp
   0x000000000040056a <+4>:     sub    rsp,0x10
   0x000000000040056e <+8>:     mov    edi,0x20
   0x0000000000400573 <+13>:    call   0x400450 <malloc@plt>
   0x0000000000400578 <+18>:    mov    QWORD PTR [rbp-0x8],rax
   0x000000000040057c <+22>:    mov    rax,QWORD PTR [rbp-0x8]
   0x0000000000400580 <+26>:    movabs rdx,0x21636f6c6c614d
   0x000000000040058a <+36>:    mov    QWORD PTR [rax],rdx
   0x000000000040058d <+39>:    mov    rax,QWORD PTR [rbp-0x8]
   0x0000000000400591 <+43>:    mov    rdi,rax
=> 0x0000000000400594 <+46>:    call   0x400430 <puts@plt>
   0x0000000000400599 <+51>:    mov    eax,0x0
   0x000000000040059e <+56>:    leave
   0x000000000040059f <+57>:    ret
End of assembler dump.
```

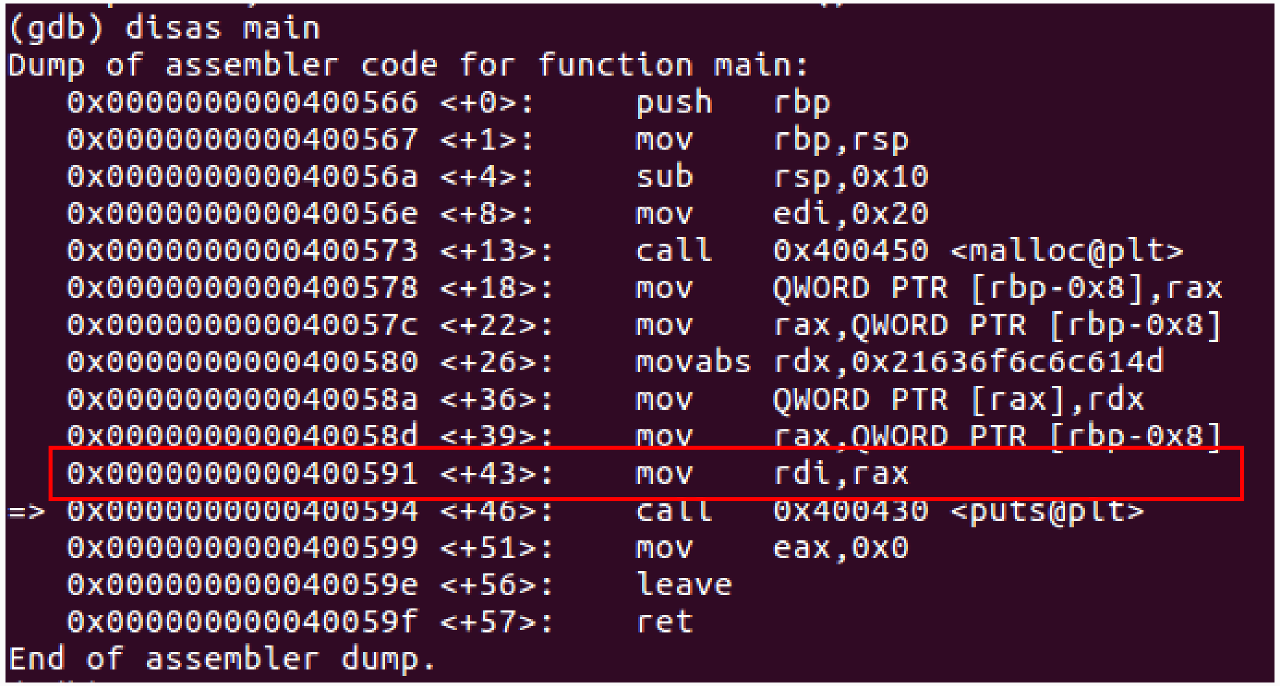


```
0x000000000040058a in main ()
(gdb) x/gx $rax
0x602010:       0x0000000000000000
(gdb) x/gx $rdx
0x21636f6c6c614d:       Cannot access me
(gdb) ni
0x000000000040058d in main ()
(gdb) x/gx $rax
0x602010:       0x0021636f6c6c614d
(gdb)
```

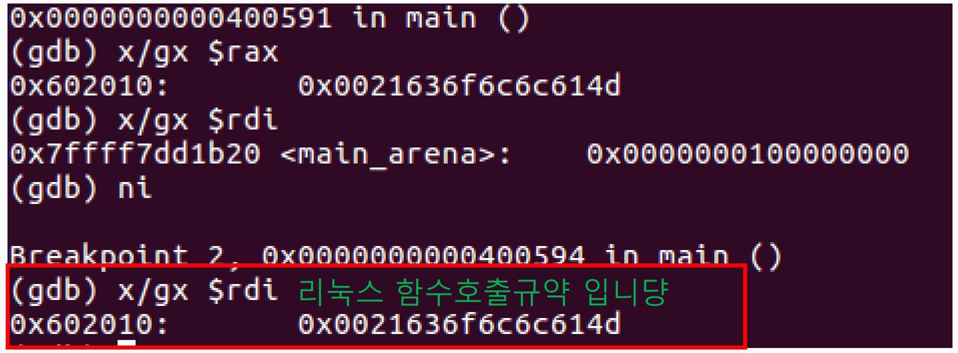


```
gdb-peda$ x/s 0x602010
0x602010:       "Malloc!"
```

메모리구조 – Heap

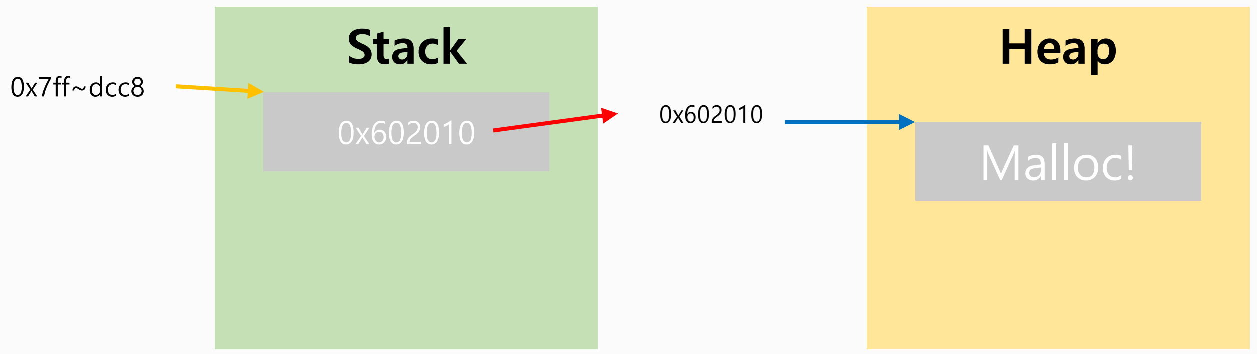


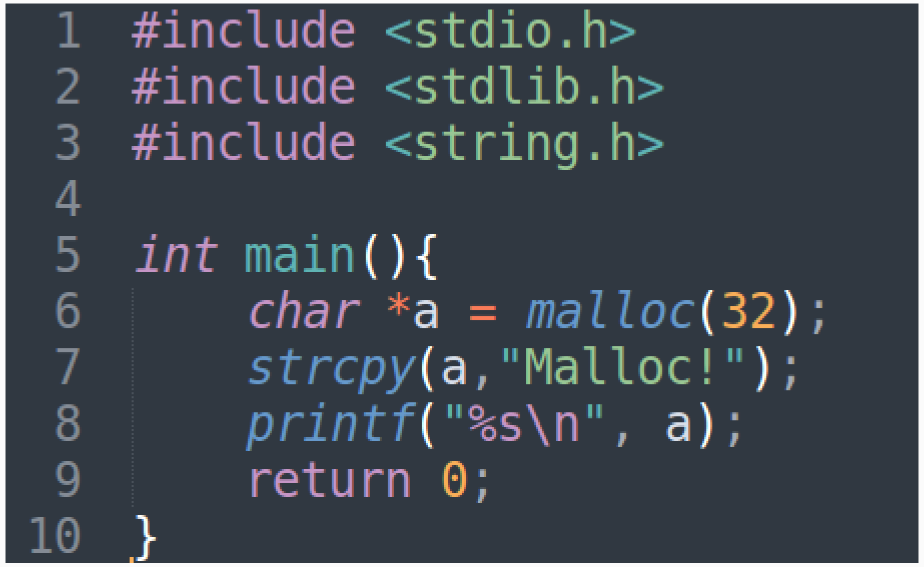
```
(gdb) disas main
Dump of assembler code for function main:
   0x0000000000400566 <+0>:     push   rbp
   0x0000000000400567 <+1>:     mov    rbp,rsp
   0x000000000040056a <+4>:     sub    rsp,0x10
   0x000000000040056e <+8>:     mov    edi,0x20
   0x0000000000400573 <+13>:    call   0x400450 <malloc@plt>
   0x0000000000400578 <+18>:    mov    QWORD PTR [rbp-0x8],rax
   0x000000000040057c <+22>:    mov    rax,QWORD PTR [rbp-0x8]
   0x0000000000400580 <+26>:    movabs rdx,0x21636f6c6c614d
   0x000000000040058a <+36>:    mov    QWORD PTR [rax],rdx
   0x000000000040058d <+39>:    mov    rax,QWORD PTR [rbp-0x8]
   0x0000000000400591 <+43>:    mov    rdi,rax
=> 0x0000000000400594 <+46>:    call   0x400430 <puts@plt>
   0x0000000000400599 <+51>:    mov    eax,0x0
   0x000000000040059e <+56>:    leave
   0x000000000040059f <+57>:    ret
End of assembler dump.
```



```
0x0000000000400591 in main ()
(gdb) x/gx $rax
0x602010:       0x0021636f6c6c614d
(gdb) x/gx $rdi
0x7ffff7dd1b20 <main_arena>:    0x0000000100000000
(gdb) ni

Breakpoint 2, 0x0000000000400594 in main ()
(gdb) x/gx $rdi 리눅스 함수호출규약 입니당
0x602010:       0x0021636f6c6c614d
```

메모리 누수(Memory Leak)



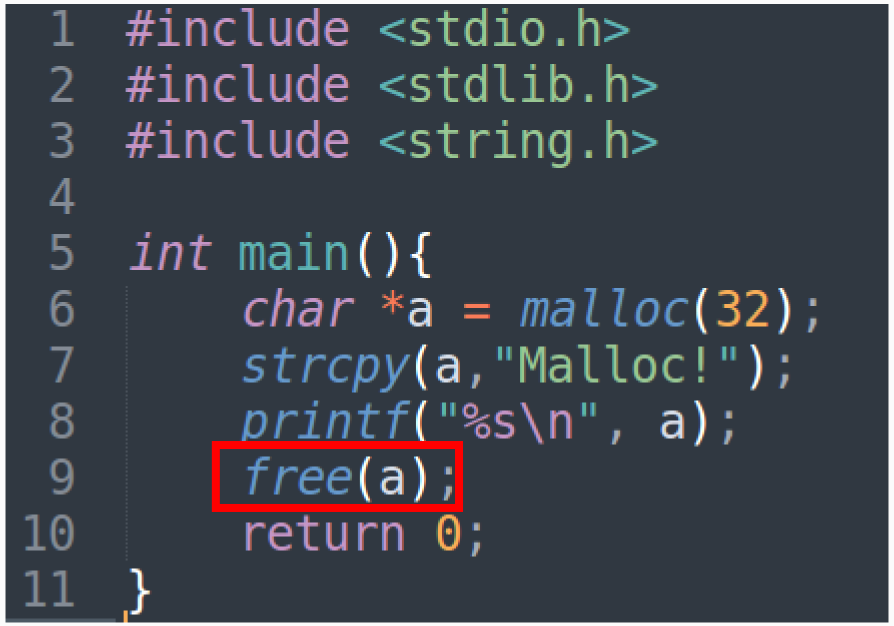


```c
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

int main(){
        char *a = malloc(32);
        strcpy(a,"Malloc!");
        printf("%s\n", a);
        return 0;
}
```

C/C++

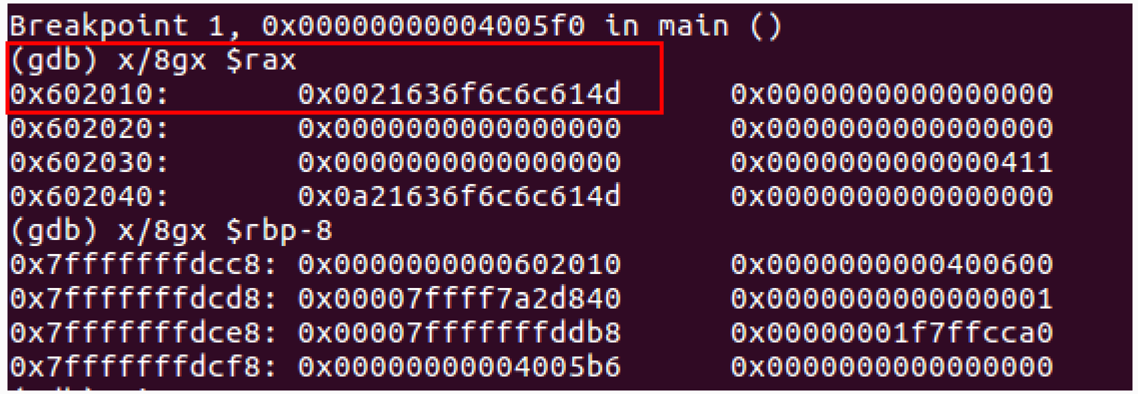
~~Garbage Collector~~ : 불필요한 메모리를 알아서 정리

Free()



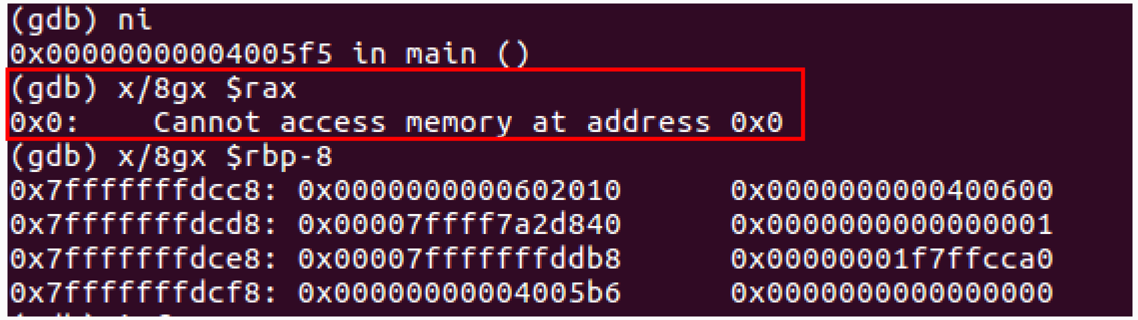
```c
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

int main(){
    char *a = malloc(32);
    strcpy(a,"Malloc!");
    printf("%s\n", a);
    free(a);
    return 0;
}
```

before



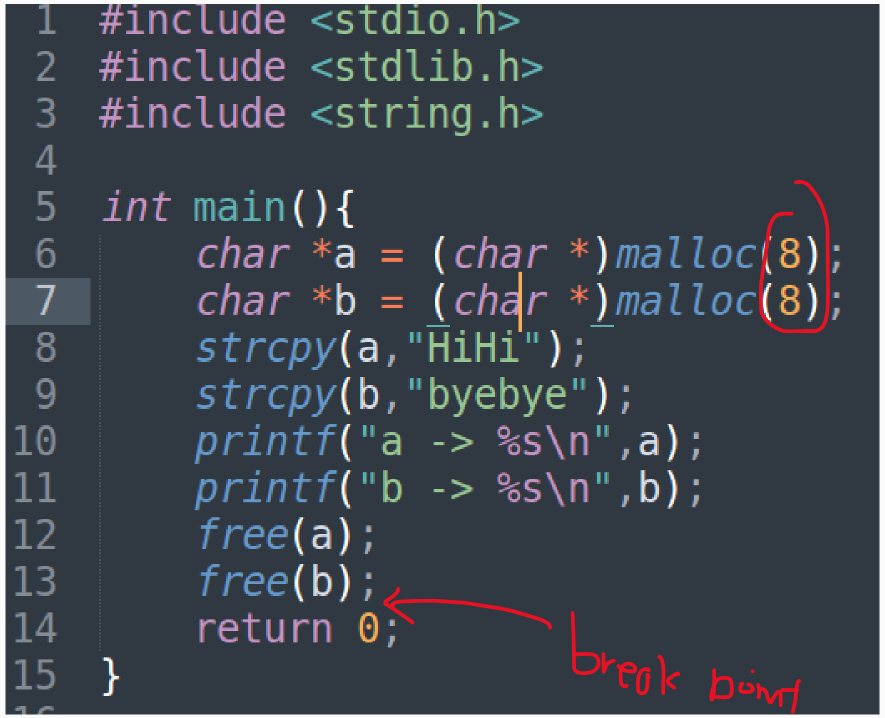
```
Breakpoint 1, 0x00000000004005f0 in main ()
(gdb) x/8gx $rax
0x602010:       0x0021636f6c6c614d      0x0000000000000000
0x602020:       0x0000000000000000      0x0000000000000000
0x602030:       0x0000000000000000      0x0000000000000411
0x602040:       0x0a21636f6c6c614d      0x0000000000000000
(gdb) x/8gx $rbp-8
0x7fffffffdcc8: 0x0000000000602010      0x0000000000400600
0x7fffffffdcd8: 0x00007ffff7a2d840      0x0000000000000001
0x7fffffffdce8: 0x00007fffffffddb8      0x00000001f7ffcca0
0x7fffffffdcf8: 0x00000000004005b6      0x0000000000000000
```

after

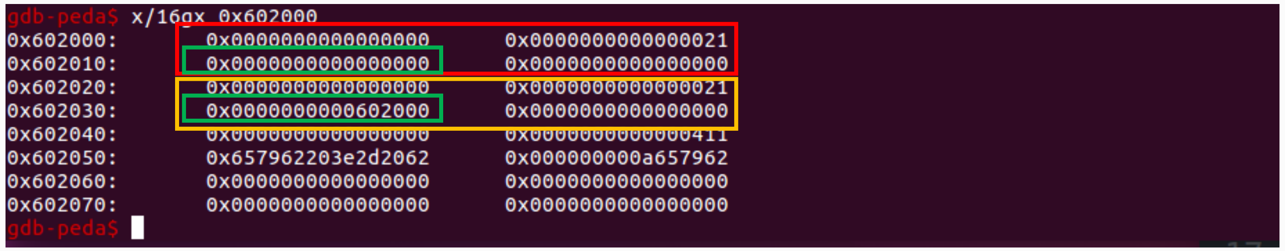


```
(gdb) ni
0x00000000004005f5 in main ()
(gdb) x/8gx $rax
0x0:    Cannot access memory at address 0x0
(gdb) x/8gx $rbp-8
0x7fffffffdcc8: 0x0000000000602010      0x0000000000400600
0x7fffffffdcd8: 0x00007ffff7a2d840      0x0000000000000001
0x7fffffffdce8: 0x00007fffffffddb8      0x00000001f7ffcca0
0x7fffffffdcf8: 0x00000000004005b6      0x0000000000000000
```

Free()



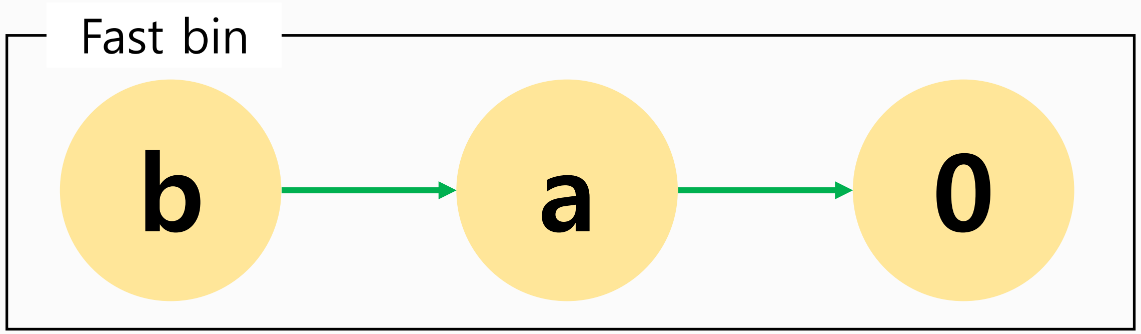
```c
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

int main(){
    char *a = (char *)malloc(8);
    char *b = (char *)malloc(8);
    strcpy(a,"HiHi");
    strcpy(b,"byebye");
    printf("a -> %s\n",a);
    printf("b -> %s\n",b);
    free(a);
    free(b);
    return 0;
}
```



```
gdb-peda$ x/16gx 0x602000
0x602000:       0x0000000000000000      0x0000000000000021
0x602010:       0x0000000000000000      0x0000000000000000
0x602020:       0x0000000000000000      0x0000000000000021
0x602030:       0x0000000000602000      0x0000000000000000
0x602040:       0x0000000000000000      0x0000000000000411
0x602050:       0x657962203e2d2062      0x000000000a657962
0x602060:       0x0000000000000000      0x0000000000000000
0x602070:       0x0000000000000000      0x0000000000000000
gdb-peda$
```

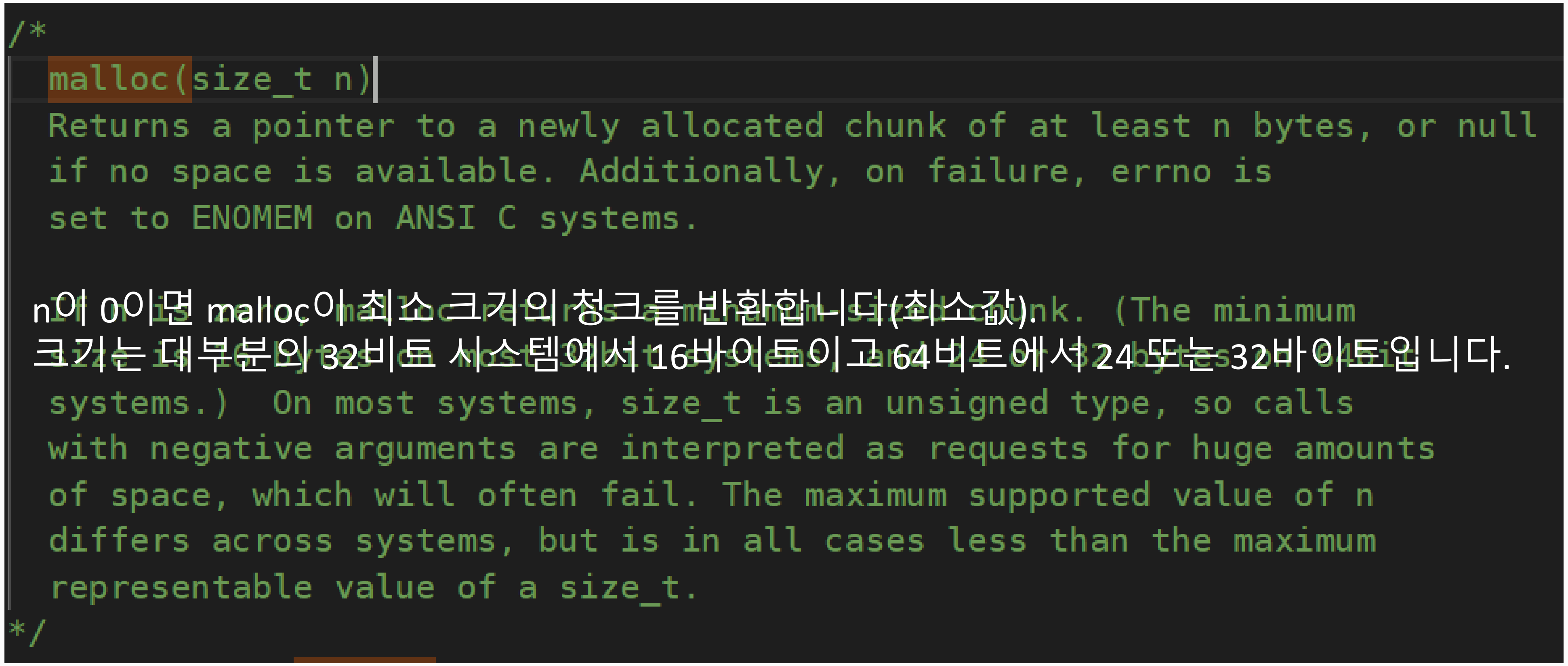
b(0x602020)->fd => 0x602000

a(0x602000)->fd => 0



$$\begin{tikzcd} b \arrow[r] & a \arrow[r] & 0 \end{tikzcd}$$

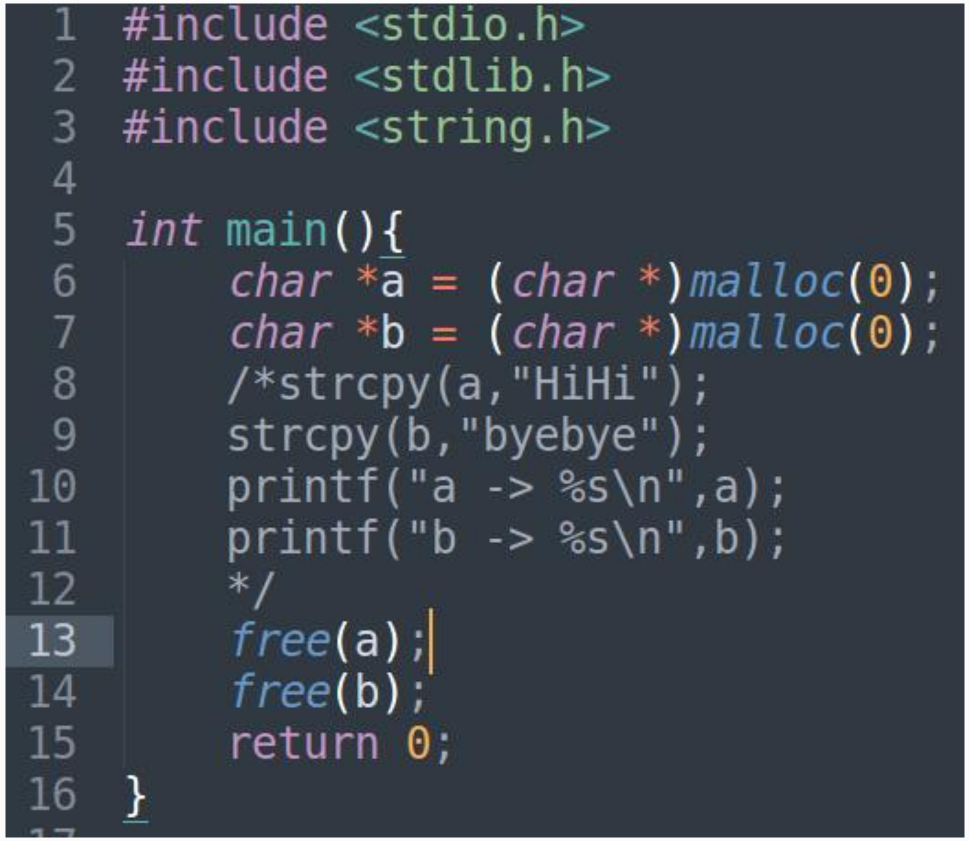
Free()



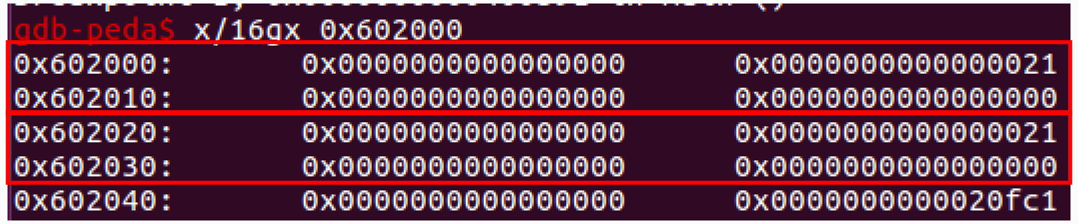
```
/*
  malloc(size_t n)
  Returns a pointer to a newly allocated chunk of at least n bytes, or null
  if no space is available. Additionally, on failure, errno is
  set to ENOMEM on ANSI C systems.

  If n is zero, malloc returns a minimum-sized chunk. (The minimum
  size is 16 bytes on most 32bit systems, and 24 or 32 bytes on 64bit
  systems.)  On most systems, size_t is an unsigned type, so calls
  with negative arguments are interpreted as requests for huge amounts
  of space, which will often fail. The maximum supported value of n
  differs across systems, but is in all cases less than the maximum
  representable value of a size_t.
*/
```

n이 0이면 malloc이 최소 크기의 청크를 반환합니다(최소값).
크기는 대부분의 32비트 시스템에서 16바이트이고 64비트에서 24 또는 32바이트입니다.

Free()

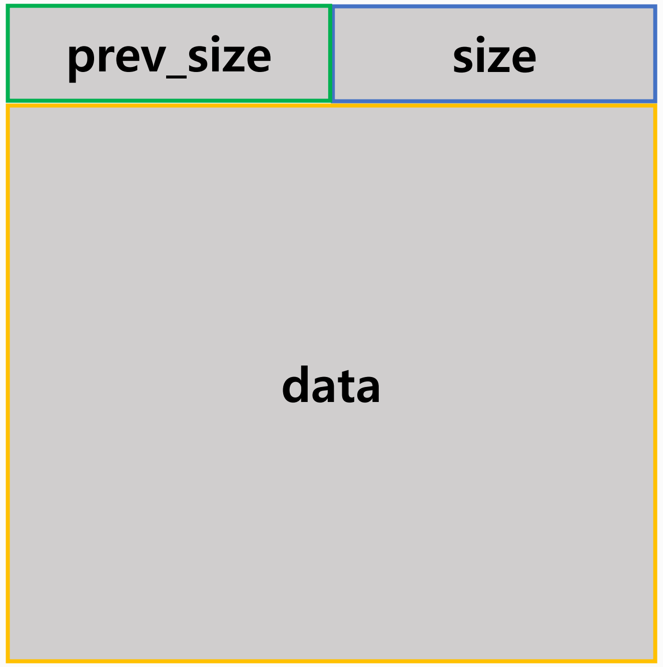


```c
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

int main(){
    char *a = (char *)malloc(0);
    char *b = (char *)malloc(0);
    /*strcpy(a,"HiHi");
    strcpy(b,"byebye");
    printf("a -> %s\n",a);
    printf("b -> %s\n",b);
    */
    free(a);
    free(b);
    return 0;
}
```

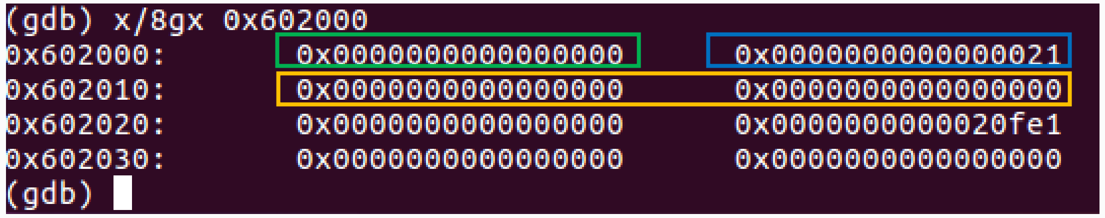


```
gdb-peda$ x/16gx 0x602000
0x602000:       0x0000000000000000      0x0000000000000021
0x602010:       0x0000000000000000      0x0000000000000000
0x602020:       0x0000000000000000      0x0000000000000021
0x602030:       0x0000000000000000      0x0000000000000000
0x602040:       0x0000000000000000      0x0000000000020fc1
```

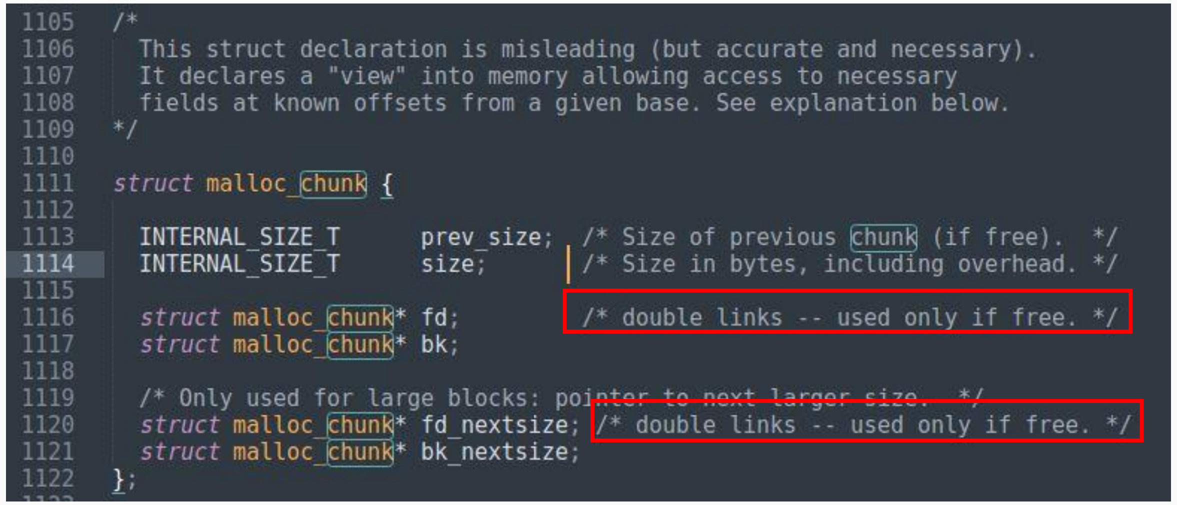
Chunk의 종류 – allocated chunk



할당된 청크
(allocated chunk)

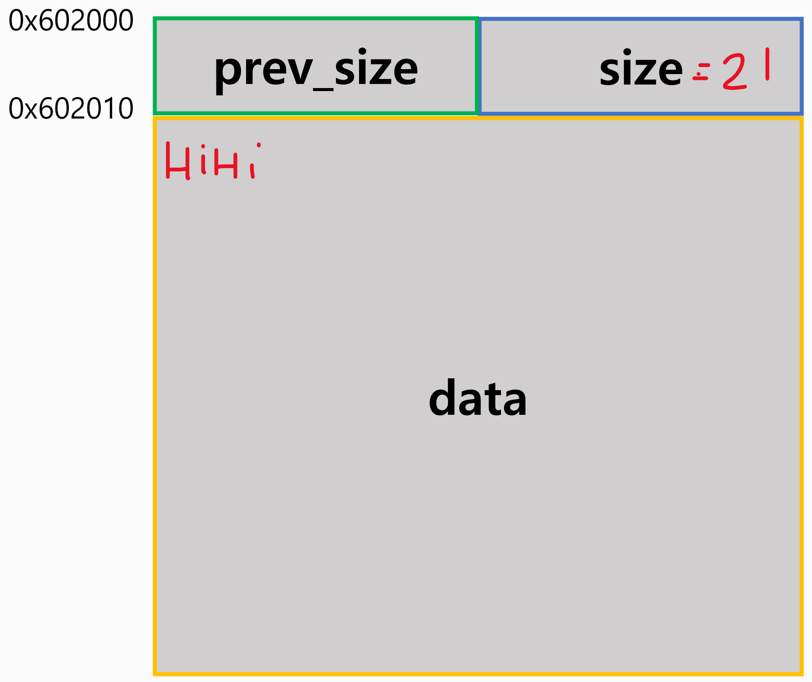


```
(gdb) x/8gx 0x602000
0x602000:       0x0000000000000000      0x0000000000000021
0x602010:       0x0000000000000000      0x0000000000000000
0x602020:       0x0000000000000000      0x0000000000020fe1
0x602030:       0x0000000000000000      0x0000000000000000
(gdb) 
```

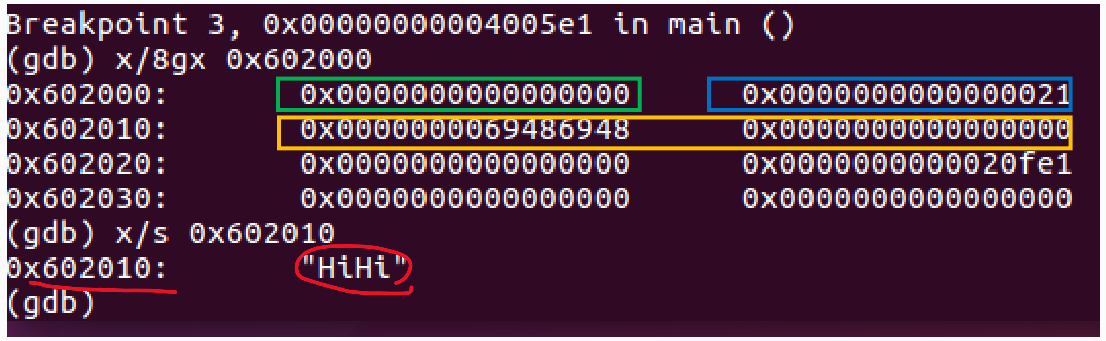


```
/*
  This struct declaration is misleading (but accurate and necessary).
  It declares a "view" into memory allowing access to necessary
  fields at known offsets from a given base. See explanation below.
*/

struct malloc_chunk {

  INTERNAL_SIZE_T      prev_size;  /* Size of previous chunk (if free).  */
  INTERNAL_SIZE_T      size;       /* Size in bytes, including overhead. */

  struct malloc_chunk* fd;         /* double links -- used only if free. */
  struct malloc_chunk* bk;

  /* Only used for large blocks: pointer to next larger size.  */
  struct malloc_chunk* fd_nextsize; /* double links -- used only if free. */
  struct malloc_chunk* bk_nextsize;
};
```

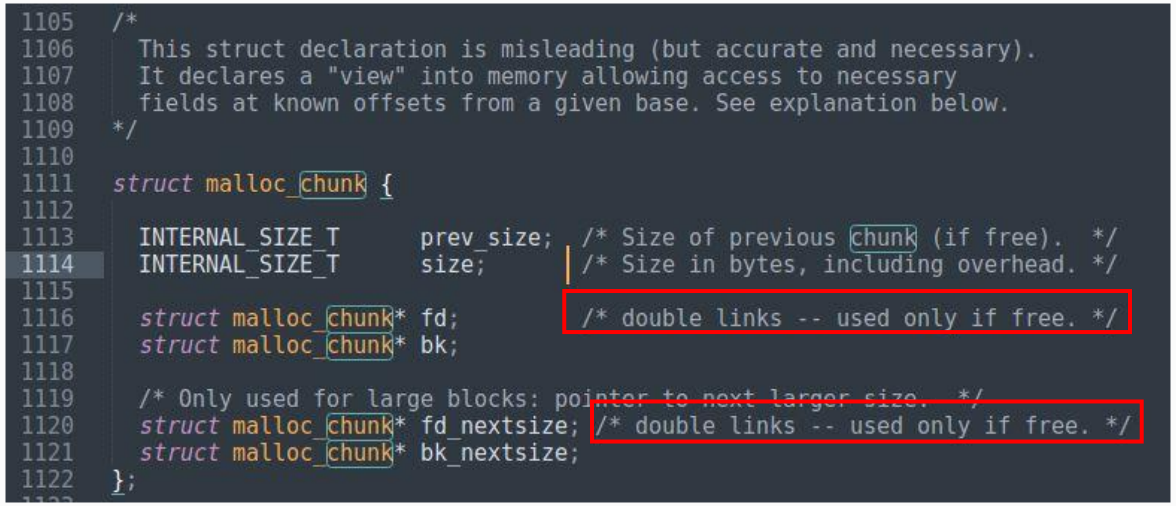
Chunk의 종류 – allocated chunk



할당된 청크
(allocated chunk)

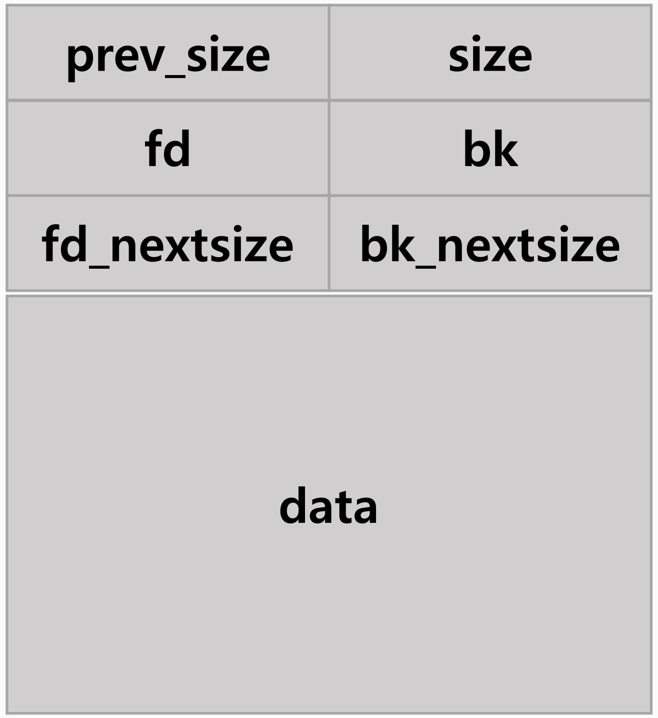


```
Breakpoint 3, 0x00000000004005e1 in main ()
(gdb) x/8gx 0x602000
0x602000:       0x0000000000000000      0x0000000000000021
0x602010:       0x0000000069486948      0x0000000000000000
0x602020:       0x0000000000000000      0x0000000000020fe1
0x602030:       0x0000000000000000      0x0000000000000000
(gdb) x/s 0x602010
0x602010:       "HiHi"
(gdb)
```

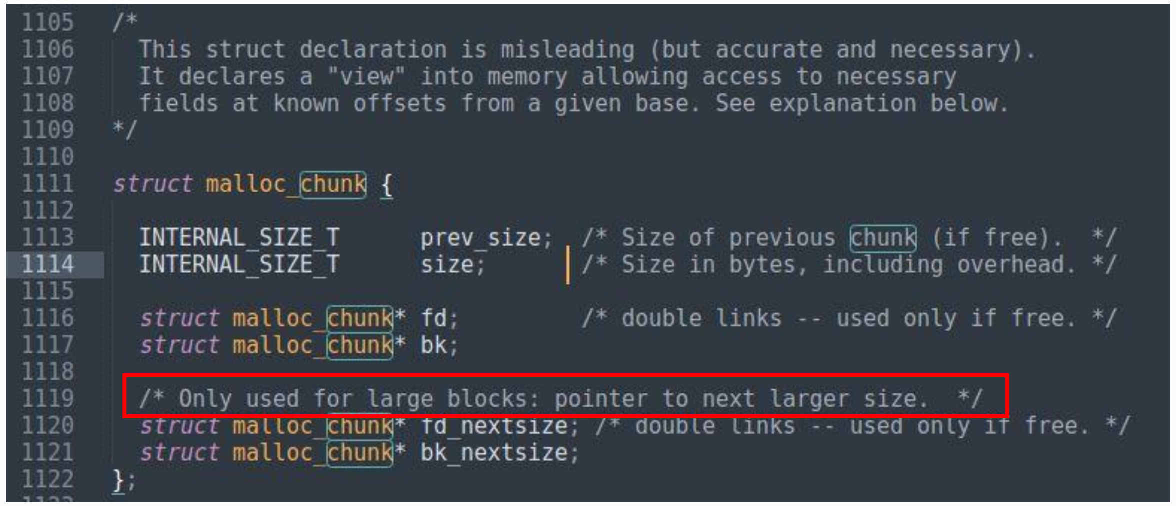


```
/*
  This struct declaration is misleading (but accurate and necessary).
  It declares a "view" into memory allowing access to necessary
  fields at known offsets from a given base. See explanation below.
*/

struct malloc_chunk {

  INTERNAL_SIZE_T      prev_size;  /* Size of previous chunk (if free).  */
  INTERNAL_SIZE_T      size;       /* Size in bytes, including overhead. */

  struct malloc_chunk* fd;         /* double links -- used only if free. */
  struct malloc_chunk* bk;

  /* Only used for large blocks: pointer to next larger size.  */
  struct malloc_chunk* fd_nextsize; /* double links -- used only if free. */
  struct malloc_chunk* bk_nextsize;
};
```

Chunk의 종류 –freed chunk



할당 해제된 청크
(free chunk)

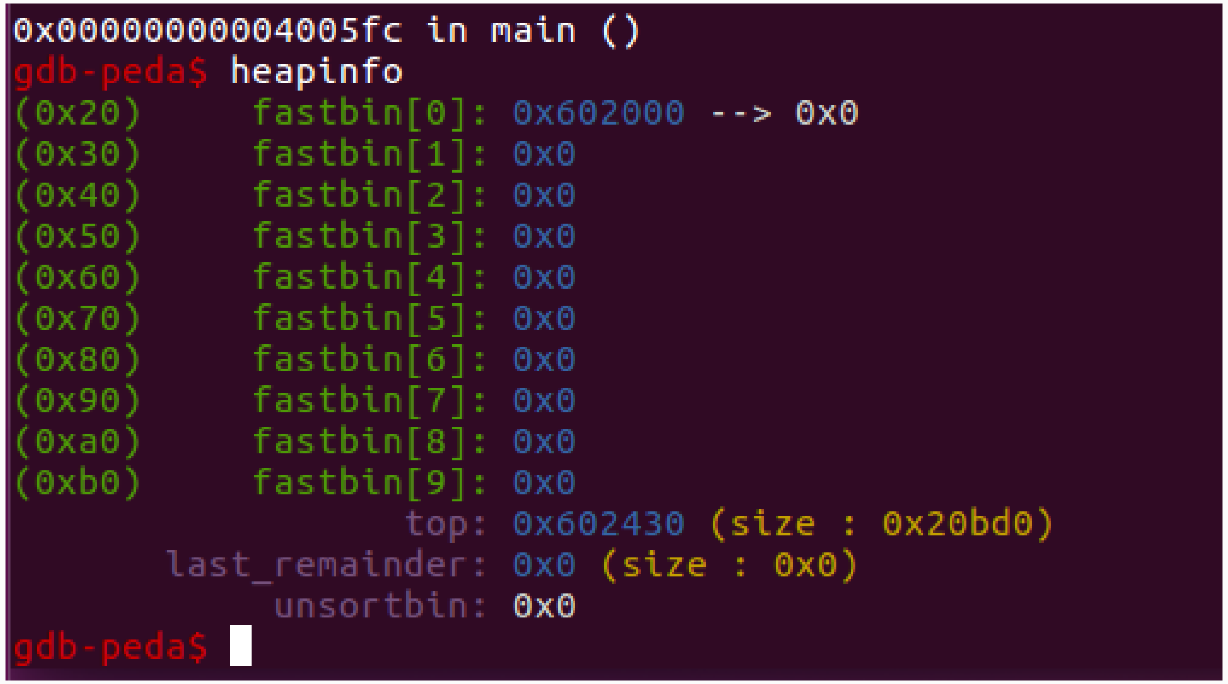


```
/*
  This struct declaration is misleading (but accurate and necessary).
  It declares a "view" into memory allowing access to necessary
  fields at known offsets from a given base. See explanation below.
*/

struct malloc_chunk {

  INTERNAL_SIZE_T      prev_size;  /* Size of previous chunk (if free).  */
  INTERNAL_SIZE_T      size;       /* Size in bytes, including overhead. */

  struct malloc_chunk* fd;         /* double links -- used only if free. */
  struct malloc_chunk* bk;

  /* Only used for large blocks: pointer to next larger size.  */
  struct malloc_chunk* fd_nextsize; /* double links -- used only if free. */
  struct malloc_chunk* bk_nextsize;
};
```

Chunk의 종류 –freed chunk

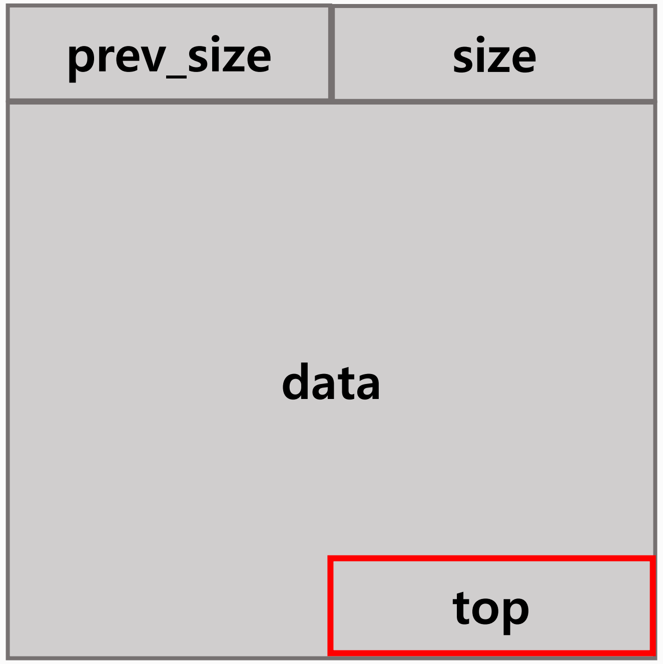
prev_size	size
fd	bk
fd_nextsize	bk_nextsize
data	

할당 해제된 청크
(freed chunk)

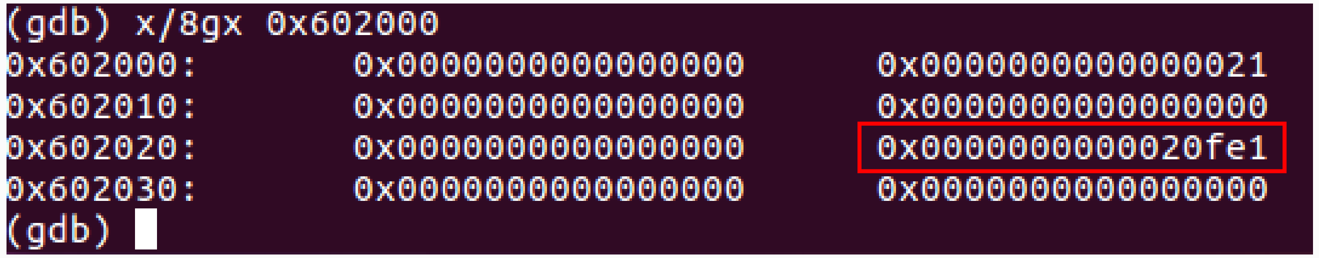


```
0x00000000004005fc in main ()
gdb-peda$ heapinfo
(0x20)     fastbin[0]: 0x602000 --> 0x0
(0x30)     fastbin[1]: 0x0
(0x40)     fastbin[2]: 0x0
(0x50)     fastbin[3]: 0x0
(0x60)     fastbin[4]: 0x0
(0x70)     fastbin[5]: 0x0
(0x80)     fastbin[6]: 0x0
(0x90)     fastbin[7]: 0x0
(0xa0)     fastbin[8]: 0x0
(0xb0)     fastbin[9]: 0x0
                  top: 0x602430 (size : 0x20bd0)
       last_remainder: 0x0 (size : 0x0)
            unsortbin: 0x0
gdb-peda$ 
```

Chunk의 종류 – top chunk



탑 청크
(top chunk)



```
(gdb) x/8gx 0x602000
0x602000:       0x0000000000000000      0x0000000000000021
0x602010:       0x0000000000000000      0x0000000000000000
0x602020:       0x0000000000000000      0x0000000000020fe1
0x602030:       0x0000000000000000      0x0000000000000000
(gdb) 
```

Arena의 가장 마지막에 위치하는 청크

새로운 malloc이 할당되면 Top Chunk에서 분리되어 청크를 할당

만약 Top Chunk에 인접한 Chunk가 free되면 Top Chunk에 병합된다.

* Arena : 각각의 스레드가 서로 간섭하지 않고 서로 다른 메모리 영역에서 액세스할 수 있게 도와주는 힙 영역

의식의 흐름 (느낀점)

Heap을 제대로 알기 위해서는 해야 할 기본 개념 공부가 많다.

많은 걸 하나씩 공부하고 있지만, 처음 공부했던 것들이 기억이 안난다.

그러다 보니 날카로운 질문이 들어올까 두렵다(?)

블로그에 정리하고 싶은데 부드럽게 이해하지 못한 부분이 많다.

아 – 갈 길이 멀다.

포너블에 입문하기엔 시간대비 습득량이 너무 적다. -> 현타가 온다.

★짱짱 SCP팀원님들! 발표 중 틀린 부분이 있으면 꼭! 꼭! 짚어주세요. 끝.