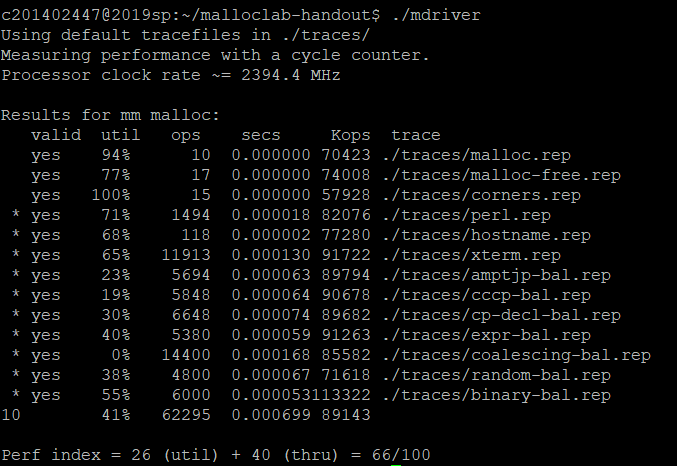
**2019시스템 프로그래밍**

**- Lab 09 -**

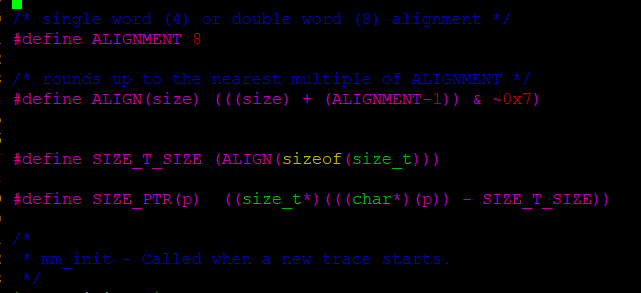
|  |  |
| --- | --- |
| **제출일자** | 2019.11.29 |
| **분 반** | 02 |
| **이 름** | 한원희 |
| **학 번** | 201402447 |

|  |
| --- |
| **Naïve** |



|  |
| --- |
| 구현 방법 |

MACRO



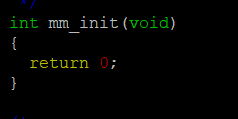
ALIGNMENT 8 : 저장 공간을 double word 사이즈 단위로 할당 함으로 8로 선언해 줍니다.

ALIGN(size) (((size) + (ALIGNMENT-1)) & ~0x7) : size의 단위를 위에 선언한 단위로 맞추어 줍니다.

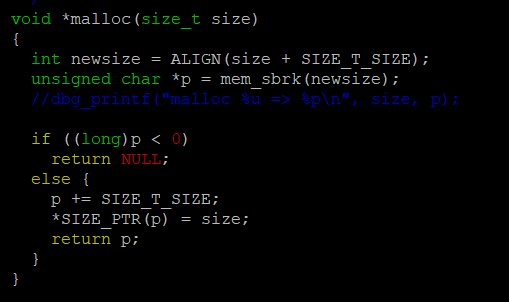
SIZE\_T\_SIZE (ALIGN(sizeof(size\_t))) : size\_t형 사이즈를 ALIGN을 이용하여 위에 선언한 단위로 맞추어 줍니다.

SIZE\_PTR(p) ((size\_t\*)(((char\*)(p)) - SIZE\_T\_SIZE)) : p에서 SIZE\_T\_SIZE 즉 8을 빼서 헤더로 이동한 다음 해당 형을 size\_t로 바꿈에 의해 앞에 4byte만 표현하여 p의 사이즈를 받아 옵니다.

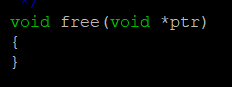
사용 함수



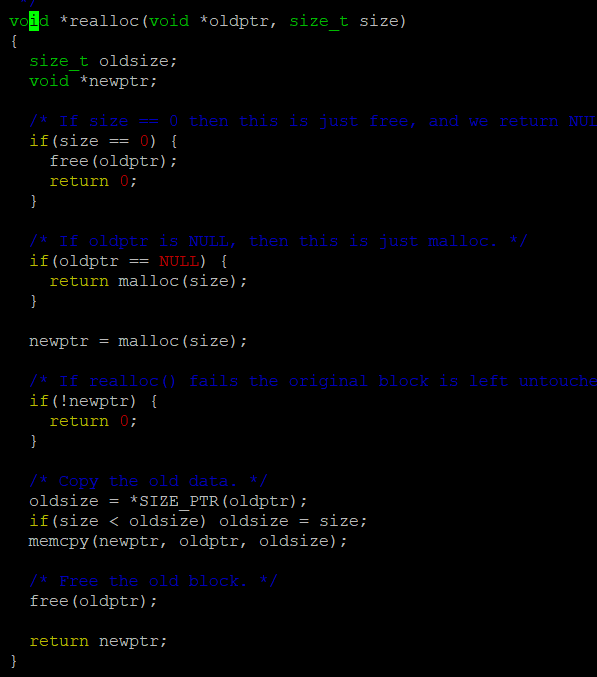
heap을 초기화 해주는 함수 구현이 되어있지 않아 0을 리턴합니다.



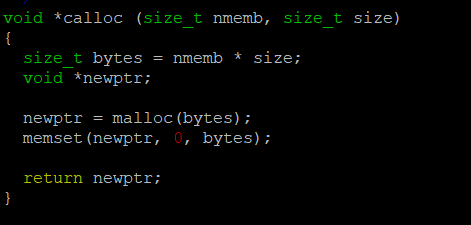
void 함수는 size바이트의 저장 공간 할당을 해주는 함수, ALIGN으로 입력받은 size를 SIZE\_T\_SIZE 즉 8을 더한 후 이 값보다 크거나 같은 8의 배수를 구하여 newsize에 넣어줍니다. 그리고 해당 newsize 만큼의 저장 공간을 p에 넣어 줍니다. 만약 p가 0 보다 작으면 즉 저장 공간을 받지 못하였으므로 NULL을 리턴 해주고 그렇지 않으면 p에 SIZE\_T\_SIZE 즉 8을 더하여 주고, p의 사이즈 값에 입력 받은 size를 넣어 줍니다. 그리고 p를 리턴 해줍니다.



할당 받은 ptr 메모리를 가용 메모리로 바꾸어 주는 함수 구현이 되어있지 않습니다.

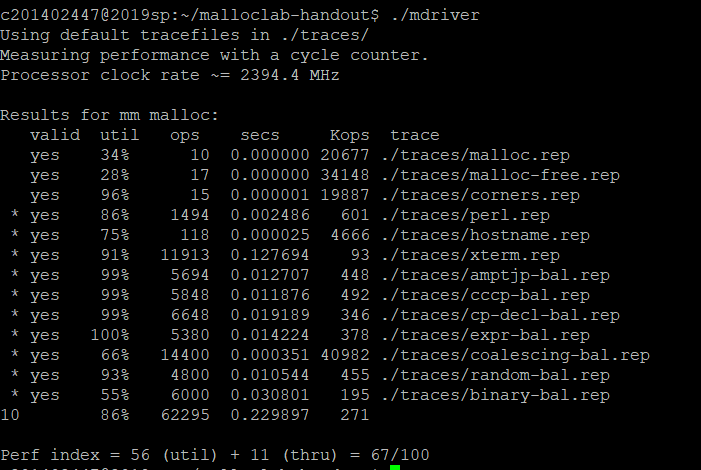


realloc 함수는 할당 받은 oldptr를 입력받은 size로 재 할당 해주는 함수입니다. size가 0이면 free와 같으므로 free 시켜준후 0리턴, oldptr이 할당 받지 않은 메모리 라면 malloc으로 입력받은 size만큼 할당해줍니다. 이 두가지를 만족 하지 못하면 newptr에 malloc으로 입력받은 size만큼 할당하고, 할당이 잘 되었는지 확인한 후 oldsize에 oldptr의 사이즈를 넣어주고, 원래의 사이즈와 다시 할당받기를 원하는 사이즈의 크기를 비교하여 작은값을 oldsize에 넣어주고 memcpy함수를 이용하여 원래 oldptr에 있었던 정보를 newptr에 oldsize만큼 복사해줍니. 즉 입력 받은 size가 작으면 입력 받은 size만큼만 복사해주고 입력 받은 size가 더 크면 oldptr의 모든 정보를 복사해줍니다. 그리고 oldptr을 free시켜준다음 다시 할당받은 newptr을 리턴해줍니다.

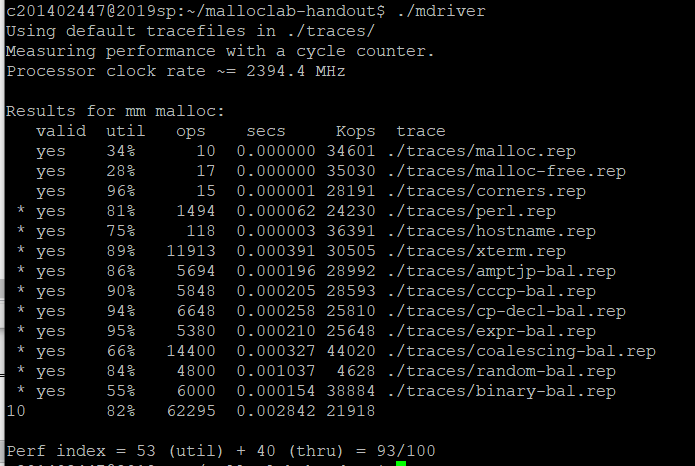


calloc 함수는 malloc과 같이 저장 공간을 할당해 주고 0으로 초기화 해주는 함수. malloc과 비슷 하지만 memset을 이용하여 0으로 모두 초기화 해줍니다.

|  |
| --- |
| **implicit** |



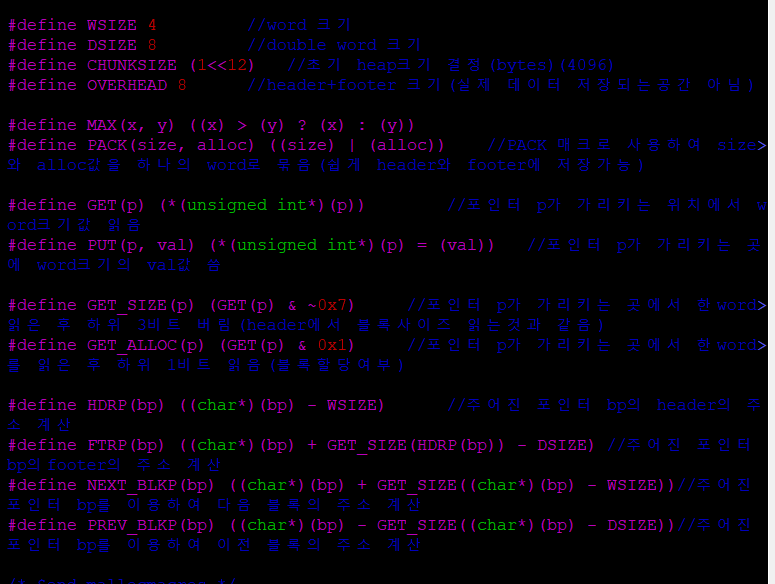
first\_fit 함수 사용



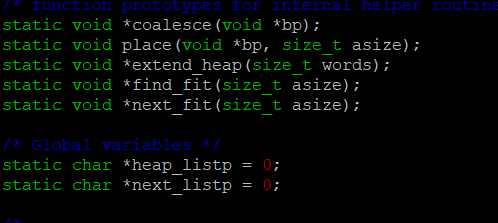
next\_fit 함수 사용

|  |
| --- |
| 구현 방법 |

MACRO

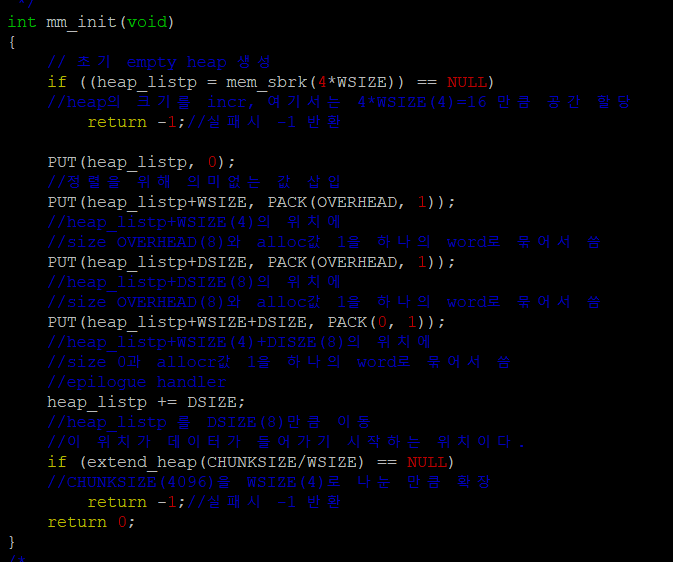


사용한 define macro는 주석을 달아두었습니다.

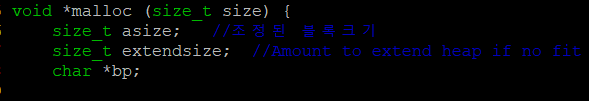


사용한 함수들을 선언해줍니다. next\_fit을 구현한 함수도 포함되어 있습니다.

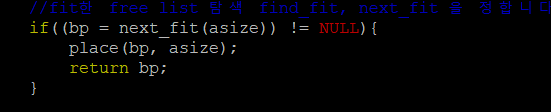
전역변수 next\_listp 도 static으로 heap\_listp 밑에 0으로 초기화 해주었습니다.



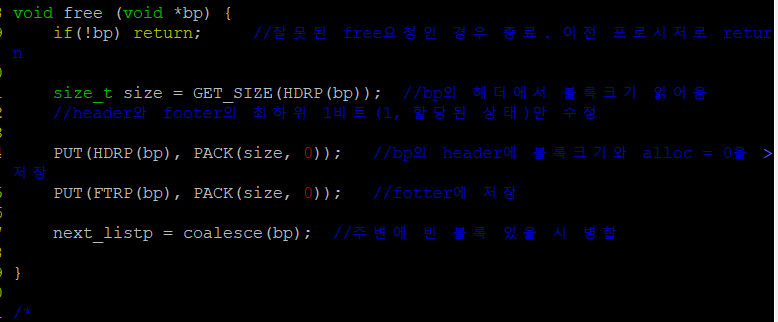
heap을 초기화 해주는 함수로 heap의 처음 블록의 헤더와 풋터를 설정해주고 free 블록을 생성해줍니다. malloc을 이용하여 요청하는 만큼의 메모리를 할당 해주기 위한 빈 저장공간을 미리 마련해 놓아줍니다.



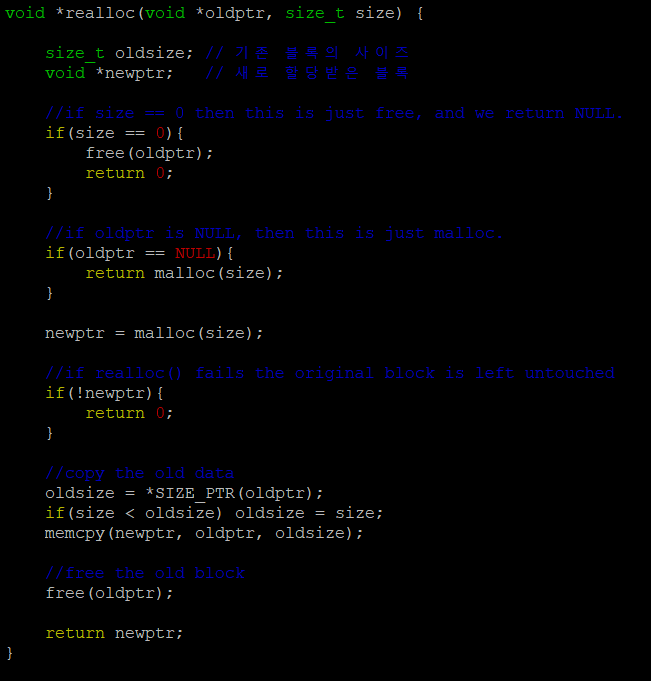
malloc 함수입니다. 위에 작성한 naive때와 같습니다. 위에서 설명을 해두었으므로 넘어가겠습니다.



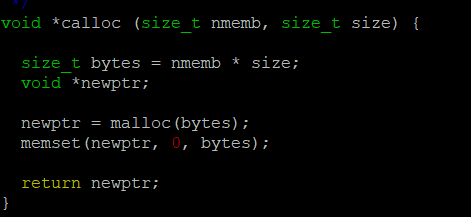
malloc 함수 중간에 next\_fit과 first\_fit 둘중에 어떤식으로 검사할지 정하는 부분이 존재합니다. next\_fit으로 정의해주었습니다.



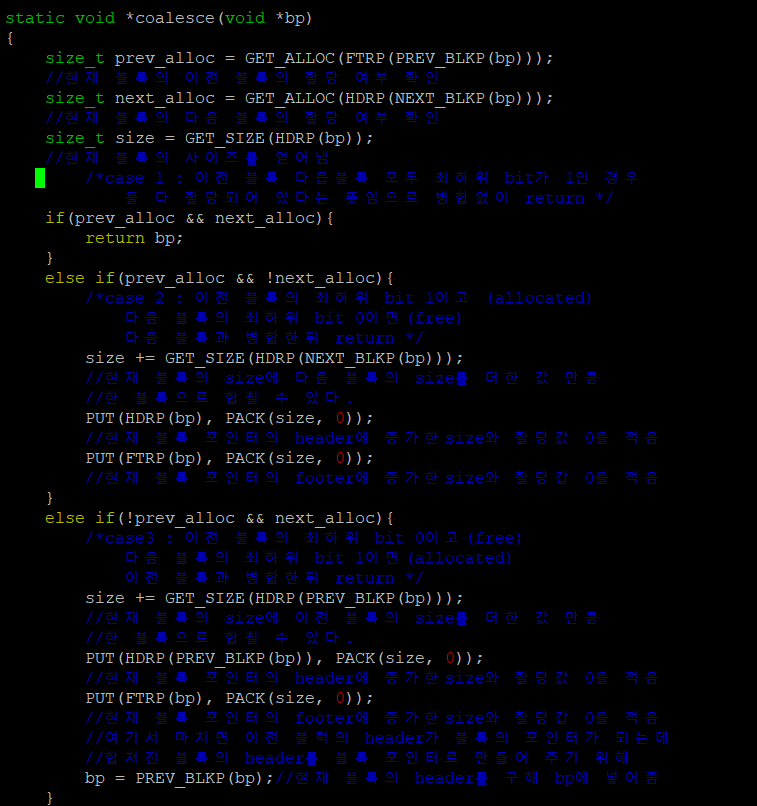
free 함수는 입력 받은 블록을 가용 메모리로 바꾸어 주는 함수로 해당 블록의 헤더와 풋터를 가용 상태로 바꾸어 주고 coalesce(연결)를 이용하여 해당 블록의 주변에 가용 블록이 있으면 정렬을 해줍니다.

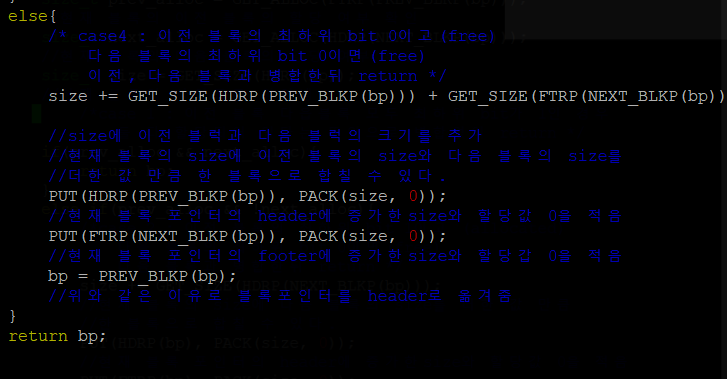


realloc 함수는 naive때 설명했으므로 넘어가겠습니다.



calloc 또한 naive에서 설명했습니다.





coalesce함수가 하는일을 보면 가장 먼저 현재 블록의 이전 블록의 풋터에서 할당여부와 현재 블록의 다음 블록의 헤더에서 할당여부를 가져와 저장합니다. 마지막으로 현재 블록 포인터로 현재 블록의 size를 알아오고나서 조건에 따른 작업을 처리합니다.

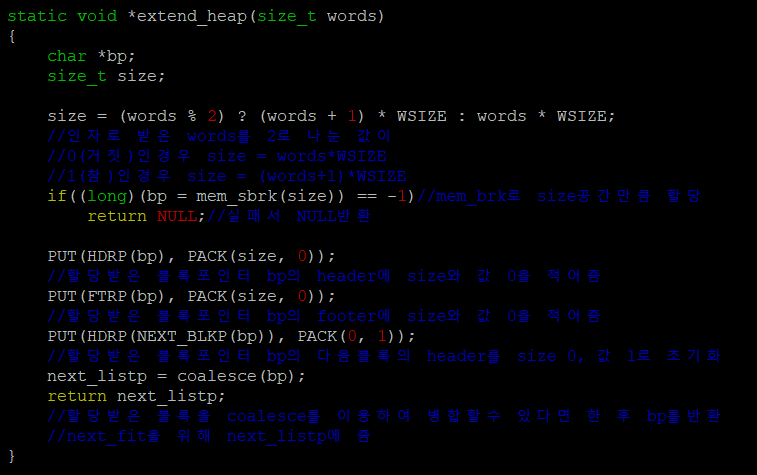
case 1을 보면 이전 블록과 다음 블록이 전부 할당되어 있는 경우로 이 때에는 현재블록과 병합 할 수 있는 블록이 없으므로 아무일도 하지않고 블록 포인터를 리턴합니다.

풋터를 옮겨줘야 할 것 같지만 풋터의 위치는 헤더의 위치에서 SIZE를 가지고 계산해 딱히 바꾸어 줄 필요가 없습니다.

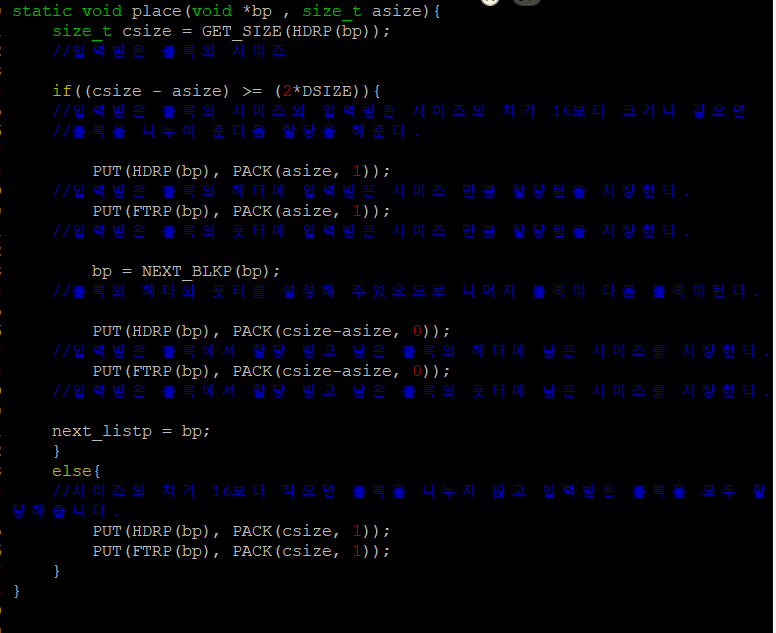
case 2를 보면 이전 블록은 할당되어있고 다음 블록은 free인 경우입니다, 이 때는 다음블록의 size를 구하여 현재 블록의 size에 더해주고 헤더와 풋터에 PUT으로 현재 블록의 더해진 size와 할당여부 0(free)을 적어줍니다.

case 3을 보면 이전 블록은 free고 다음 블록은 할당되어있는 경우입니다, 이때는 이전 블록의 size를 구하여 현재 블록의 size에 더해주고 헤더와 풋터에 PUT으로 현재 블록에 더해진 size와 할당여부 0(free)을 적어줍니다. 또한 헤더앞에 블록이 병합되어 헤더의 위치가 변해야 함으로 블록포인터 bp에 현재 블록의 헤더를 다시 구하여 저장해줍니다.

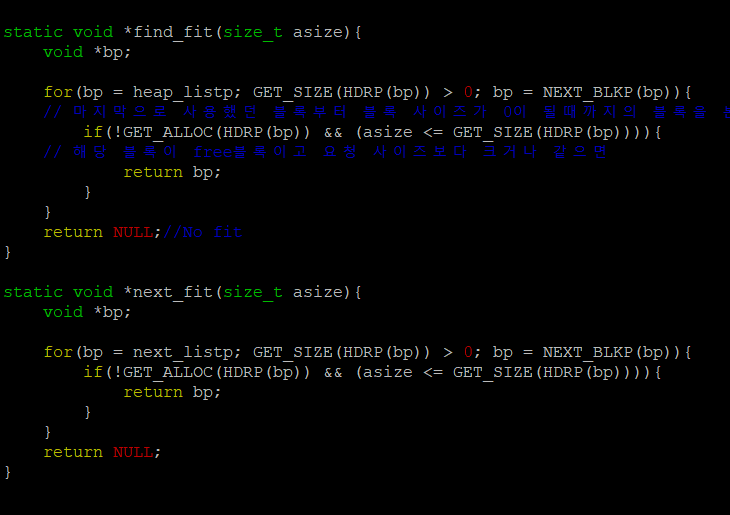
case 4를 보면 이전 블록과 다음 블록 모두 free인 경우입니다. 이때는 이전 블록과 다음블록의 size를 전부 구하여 현재 블록의 size와 더해 주고 헤더와 풋터에 PUT으로 현재 블록에 더해진 size와 할당여부 0(free)을 적어줍니다. 또한 헤더앞에 블록이 병합되어 헤더의 위치가 변해야 함으로 블록포인터 bp에 현재 블록의 헤더를 다시 구하여 저장해줍니다. 끝으로 블록 포인터를 반환해줍니다.



extend\_heap 함수는 메모리에 더 이상 할당할 수 있는 블록을 찾을 수 없을 때 추가적으로 mem\_sbrk함수로 메모리를 할당 받기 위하여 사용 합니다. 맨 처음 블록의 할당조건을 만족하기 위하여 words를 짝수로 만들어주기 위하여 홀수인 경우 1을 더해 계산하고, 짝수인 경우 그대로 계산해줍니다. words를 짝수로 바꾸어 준 후에 WSIZE(4)를 곱하고, 이 값 만큼의 메모리를 할당받습니다. 할당받은 메모리 블록이 free상태라는 것을 표시하기 위하여 PUT으로 Header와 Footer에 size와 할당값 0(free)를 적어줍니다. 그리고 인접한 블록중 free블록이 있을 가능성이 있으므로 coalesce함수를 사용하여 병합할 수 있다면 병합한후, 병합한 새 free블록의 주소값으로 next\_listp를 옮겨줍니다.



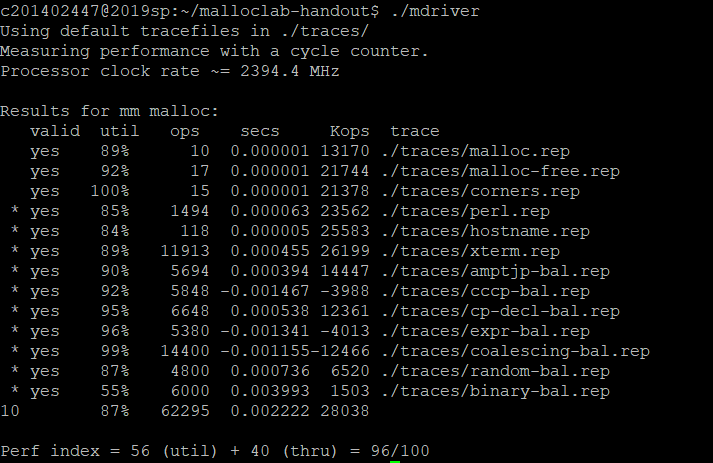
place 함수는 malloc 에서 사용되며 인자로 받은 블록 포인터에 asize만큼 할당을 해 줍니다. 할당을 하기에 앞서 블록에 asize의 공간을 할당했을 때 할당할 블록의 남은 공간에 다른 블록을 할당할 수 있는지 여부를 검사하게 됩니다. 이때 검사하는 방법은 할당한 블록 csize와 현재 할당해야 할 asize의 차가 한 블록의 최소 크기인 DSIZE+OVERHEAD즉 16보다 큰지를 검사하면 됩니다. 만약 16보다 큰 경우는 할당하고 남은 공간에 다른 블록을 할당 할 수 있음으로 블록중 asize의 크기는 할당을 하고 나머지 csize-asize는 새로운 free블록으로 만들어 줍니다. 반면 csize와 asize의 차가 16보다 작은 경우는 남는 공간에 한 개의 블록도 할당할 수 없음으로 free할 때 편하도록 csize를 전부 할당하여 줍니다.



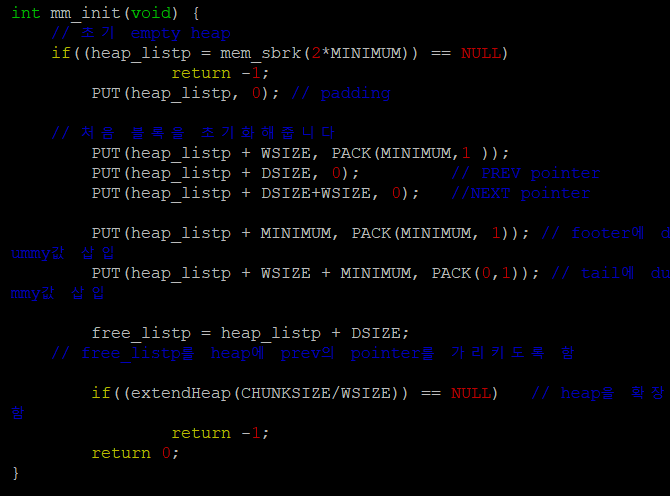
find\_fit과 next\_fit 함수입니다. first\_fit과 next\_fit 두가지 방식을 구현했습니다.

first\_fit은 입력된 사이즈에 맞는 빈 블록을 찾아 처음부터 순차 탐색하는 방식이고, next\_fit은 first\_fit과 유사하지만, 제일 마지막으로 탐색한 지점에서부터 순차탐색을 시행하는 방식입니다.

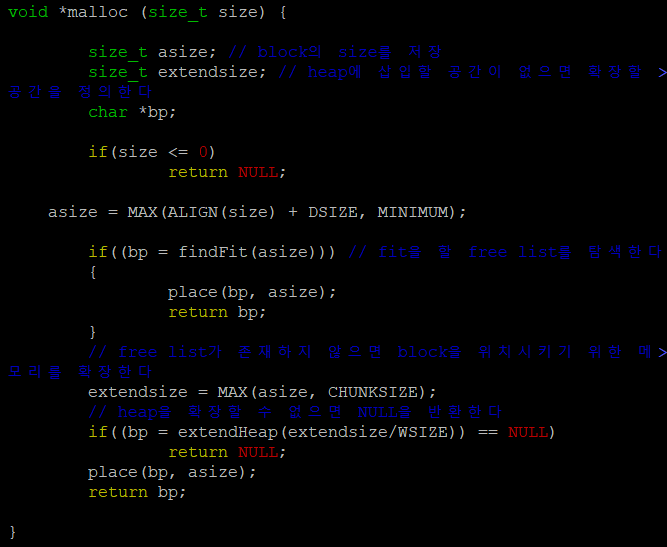
|  |
| --- |
| **explicit** |



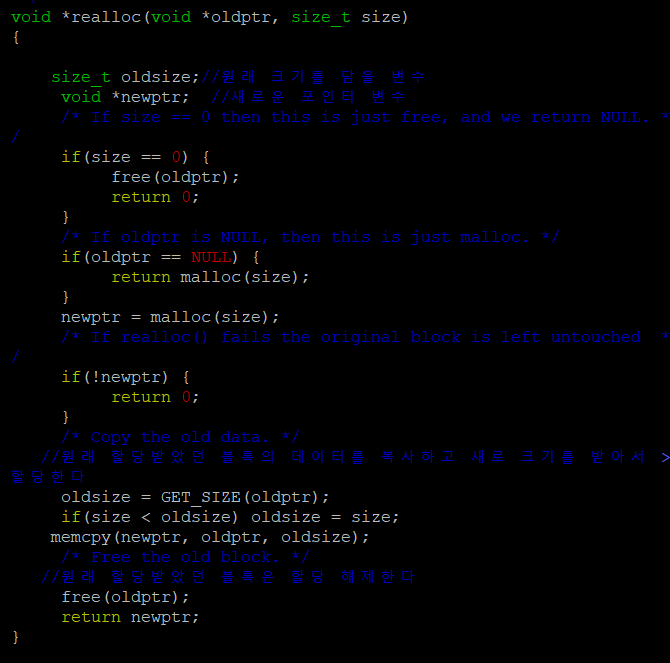
|  |
| --- |
| 구현 방법 |



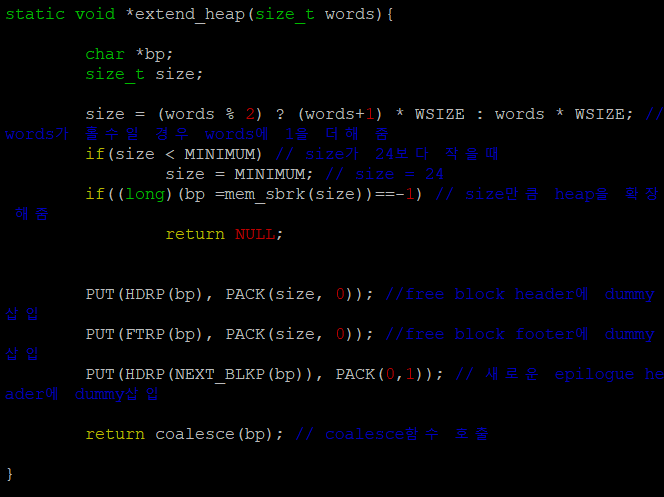
init 함수입니다. 맨 처음은 패딩으로 지정해주고 그 다음 최소 블록 크기(MINIMUM = 24바이트)를 갖는 블록이 오도록 구현해줍니다. 이 블록은 가용 리스트의 끝 부분으로 구현됩니다. 가용 블록들은 리스트의 앞 쪽에 계속 삽입되므로, 가용 리스트의 제일 마지막은 최소 블록 크기로 할당된 이 초기화된 블록이 옵니다. static 전역 변수 free\_listp는 항상 가용 리스트의 앞부분(루트)을 가리키도록 해줍니다. 이 포인터와 위의 초기 힙 블록 사이에 가용 블록들이 리스트로 연결됩니다. 힙의 마지막 부분에는 크기가 0인 할당 블록으로 에필로그를 구현한다.



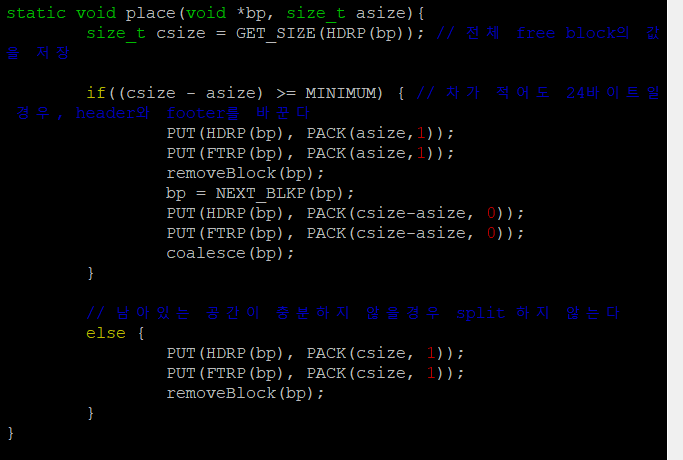
ppt에 적혀 있는 부분으로 size가 올바르지 않을때의 예외 처리 부분인 size <= 0 일때를 추가해 주었습니다.



realloc 부분은 implicit과 동일합니다.



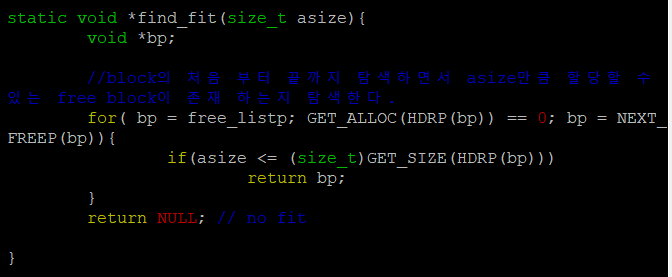
extend\_heap 함수입니다. 메모리에 추가로 힙 공간을 요구하는 함수입니다. 정렬을 유지하기 위해 요구되는 힙 크기를 짝수로 만들어준 이후 sbrk를 이용하여 메모리에 힙 공간을 요구해줍니다. 새롭게 할당된 공간을 가용 공간으로 만들어 줍니다. 이전에 존재하던 에필로그 부분이 새롭게 할당된 가용공간의 헤더 부분이 됩니다. 그리고 새롭게 생긴 가용 공간에 에필로그를 새로 정해줍니다. 리턴은 coalesce 함수를 호출하면서 반환됩니다. 이는 새롭게 요구받은 가용 공간 이전 블록이 가용 블록이었다면 합병을 해주고 가용 리스트에 추가해줍니다.



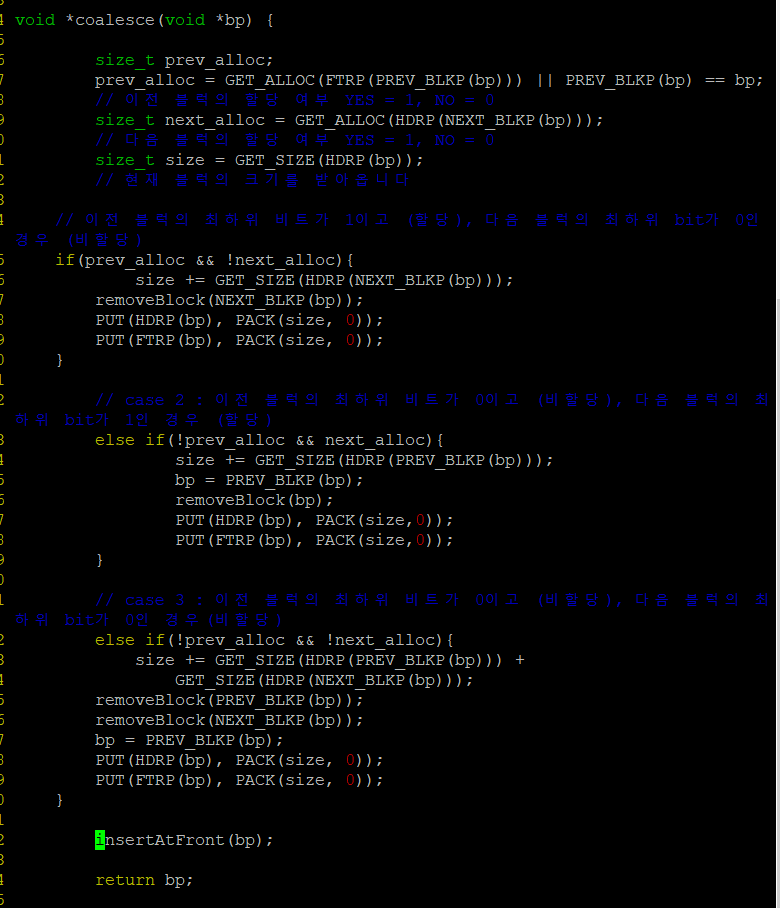
place 함수 입니다.

가용 블록의 크기와 배치하려는 블록의 크기의 차이가 최소 블록 크기(24 바이트) 이상이라면, 블록을 할당하고 분할을 수행하도록 합니다. 해당하는 크기의 블록을 할당하고, 이 블록을 가용 리스트에서 제거합니다 그리고 남는 공간을 가용 블록으로 만들어 주기 위해 헤더와 풋터를 설정해주고, coalesce를 호출합니다.

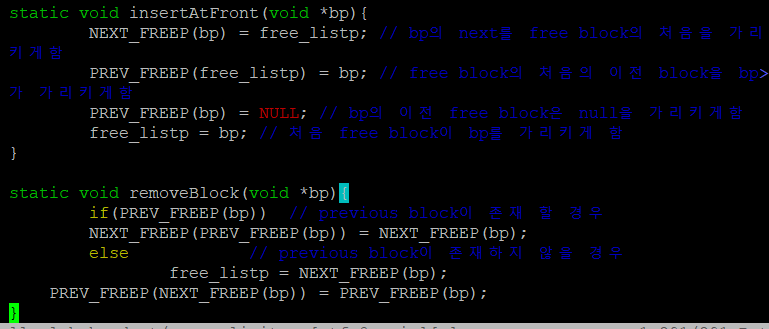
만약 할당하고 남는 부분이 최소 블록 크기가 되지 않으면 분할을 하지 않고, 블록을 할당하고, 할당 받은 블록을 가용리스트에서 제거해 줍니다.



find\_fit 함수입니다. implicit에서는 마지막 free블럭부터 검색을 하여 알맞은 블록을 찾았지만 여기서는 free블럭을 LIFO방식으로 관리해주기 때문에 free list root부터 검색을 해주는 LIFO방식입니다.



coalesce 함수입니다. free된 블록을 입력받아 해당 블록의 앞과 뒤에 있는 블록이 free인지에 따라 free인 블록이 있으면 해당 블록과 합쳐주는 함수입니다. implicit과 달리 블록에서 다음 블록과 이전 블록을 가리키는 주소도 수정을 해 주어야 합니다.



insertAtFront 함수와 removeBlock 함수입니다.

자료구조의 더블 링크드 리스트와 비슷합니다. 즉, 연결 구조를 바꾸어서 블록을 제거, 삽입할 수 있습니다.