Problem 65. (10 points):

Consider an allocator that uses an implicit free list. Each memory block, either allocated or free, has a size that is a multiple of eight bytes. Thus, only the 29 higher order bits in the header and footer are needed to record block size, which includes the header and footer and is represented in units of bytes. The usage of the remaining 3 lower order bits is as follows:

- `bit 0` indicates the use of the current block: 1 for allocated, 0 for free.
- `bit 1` indicates the use of the previous adjacent block: 1 for allocated, 0 for free.
- `bit 2` is unused and is always set to be 0.

Five helper routines are defined to facilitate the implementation of `free(void *p)`. The functionality of each routine is explained in the comment above the function definition. Fill in the body of the helper routines the code section label that implement the corresponding functionality correctly.

```
/* given a pointer p to an allocated block, i.e., p is a
   pointer returned by some previous malloc()/realloc() call;
   returns the pointer to the header of the block*/
void * header(void* p)
{
  void *ptr;

  _______;
  return ptr;
}

A. ptr=p-1
B. ptr=(void *)((int *)p-1)
C. ptr=(void *)((int *)p-4)


/* given a pointer to a valid block header or footer,
   returns the size of the block */
int size(void *hp)
{
  int result;

  _______;
  return result;
}

A. result=(*hp)&(~7)
B. result=((*(char *)hp)&(~5))<<2
C. result=(*(int *)hp)&(~7)
```

```
/* given a pointer p to an allocated block,i.e. p is
   a pointer returned by some previous malloc()/realloc() call;
   returns the pointer to the footer of the block*/
void * footer(void *p)
{
  void *ptr;

  _______;
  return ptr;
}
```

A. ptr=p+size(header(p))-8
B. ptr=p+size(header(p))-4
C. ptr=(int *)p+size(header(p))-2

```
/* given a pointer to a valid block header or footer,
   returns the usage of the currect block,
   1 for allocated, 0 for free */
int allocated(void *hp)
{
  int result;

  ______;
  return result;
}
```

A. result=(*(int *)hp)&1
B. result=(*(int *hp)&0
C. result=(*(int *)hp)|1

```
/* given a pointer to a valid block header,
   returns the pointer to the header of previous block in memory */
void * prev(void *hp)
{
  void *ptr;

  ______;
  return ptr;
}
```

A. ptr = hp - size(hp)
B. ptr = hp - size(hp-4)
C. ptr = hp - size(hp-4) + 4