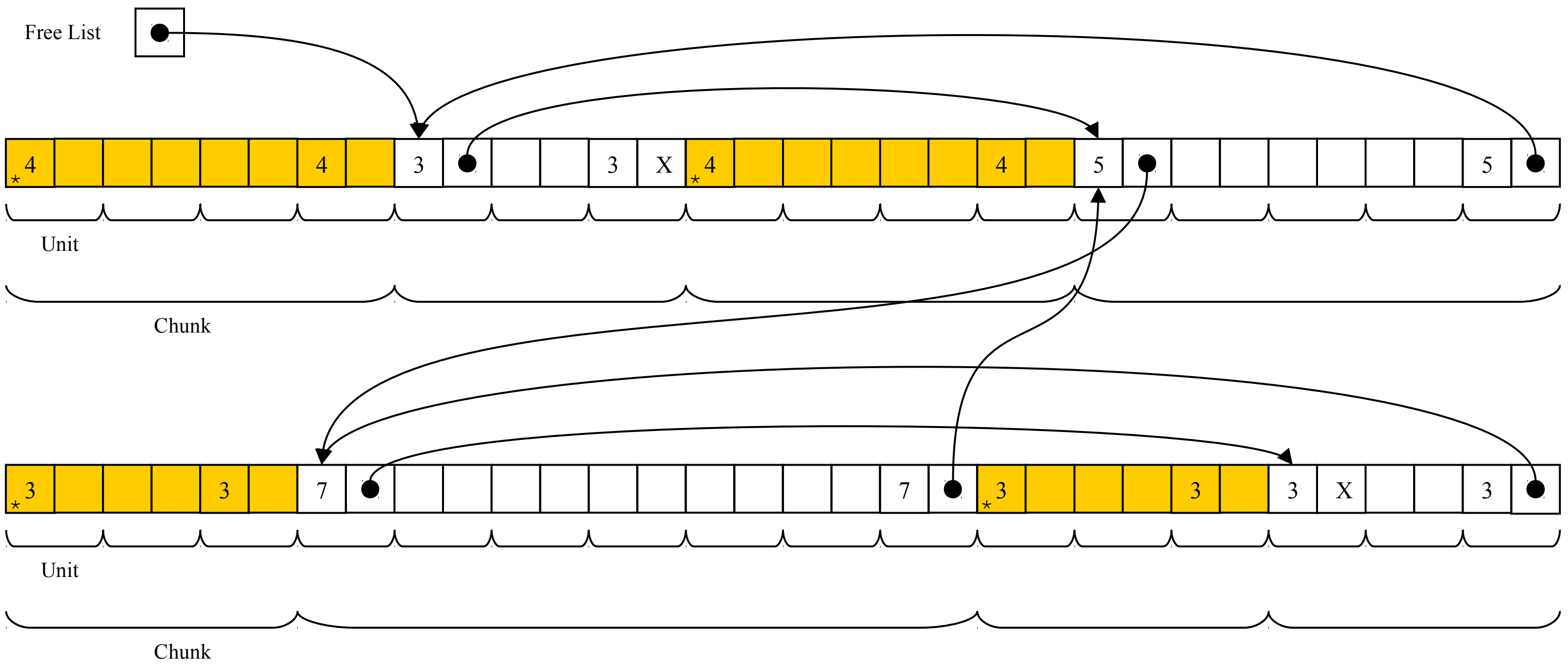
Princeton University
COS 217: Introduction to Programming Systems
HeapMgr4 Data Structures



Each box consists of 8 bytes.
Each Chunk's header Unit contains a status (inuse or free), a length, and, if the Chunk is free, a pointer to the next Chunk in the Free List.
Each Chunk's footer Unit contains a length and, if the Chunk is free, a pointer to the previous Chunk in the Free List.
The Chunks in the Free List are in no particular order.
* means INUSE; absence of * means FREE.

Copyright © 2018 by Robert M. Dondero, Jr.

Princeton University
COS 217: Introduction to Programming Systems
HeapMgr4 Algorithms

void *HeapMgr_malloc(size_t uBytes)

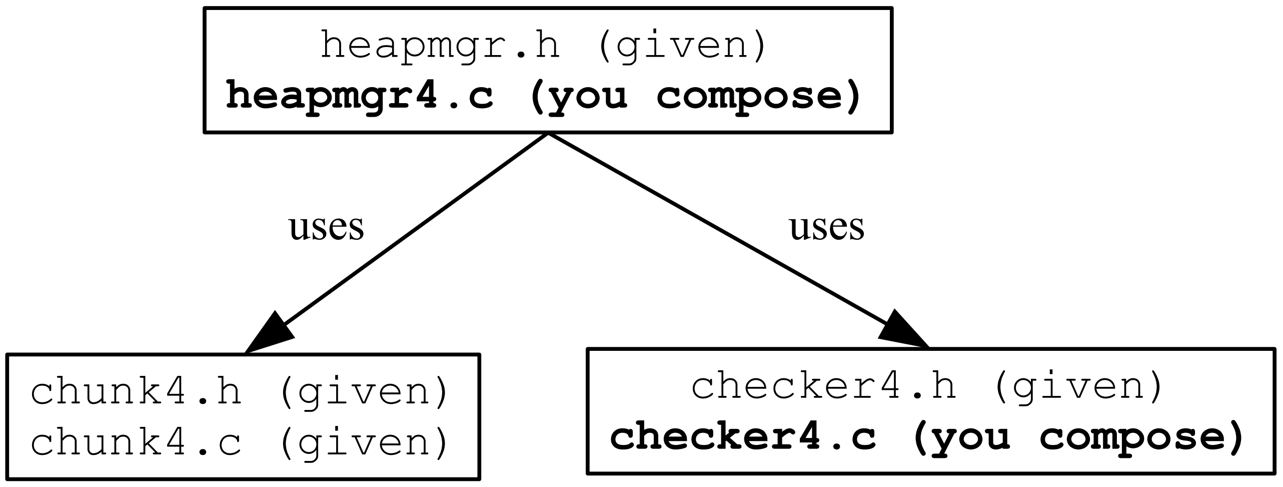
```
(1) If this is the first call of HeapMgr_malloc(), then initialize the heap manager.

(2) Determine the number of units the new chunk should contain.

(3) For each chunk in the free list...

        If the current free list chunk is big enough...

                If the current free list chunk is close to the requested size, then remove
                it from the free list, set its status to INUSE, and return it.  If the
                current free list chunk is too big, then remove it from the free list,
                split the chunk, insert the tail end of it into the free list at the front,
                set the status of the front end of it to INUSE, set the status of the tail
                end of it to FREE, and return the front end of it.

(4) Ask the OS for more memory — enough for the new chunk.  Return NULL if the OS refuses.
    Create a new chunk using that memory.  Insert the new chunk into the free list at the
    front.  If appropriate, coalesce the new chunk and the previous one in memory.  To do
    so, remove the current chunk from the free list, remove the previous chunk in memory
    from the free list, coalesce them to form a larger chunk, and insert the larger chunk
    into the free list at the front.  Let the current free list chunk be the new chunk.

(5) If the current free list chunk is close to the requested size, then remove it from the
    free list, set its status to INUSE, and return it.  If the current free list chunk is
    too big, then remove it from the free list, split the chunk, insert the tail end of it
    into the free list at the front, set the status of the front end of it to INUSE, set
    the status of the tail end of it to FREE, and return the front end of it.
```

void HeapMgr_free(void *pv)

```
(1) Set the status of the given chunk to FREE.

(2) Insert the given chunk into the free list at the front.

(3) If appropriate, coalesce the given chunk and the next one in memory.  To do so, remove
    the given chunk from the free list, remove the next chunk in memory from the free
    list, coalesce them to form a larger chunk, and insert the larger chunk into the free
    list at the front.

(4) If appropriate, coalesce the given chunk and the previous one in memory.  To do so,
    remove the given chunk from the free list, remove the previous chunk in memory from
    the free list, coalesce them to form a larger chunk, and insert the larger chunk into
    the free list at the front.
```

Copyright © 2018 by Robert M. Dondero, Jr.

Princeton University
COS 217: Introduction to Programming Systems
HeapMgr4 Code



Copyright © 2018 by Robert M. Dondero, Jr.

chunk4.h (Page 1 of 3)

```
/*--------------------------------------------------------------------*/
/* chunk4.h                                                           */
/* Author: Bob Dondero                                                */
/*--------------------------------------------------------------------*/

#ifndef CHUNK4_INCLUDED
#define CHUNK4_INCLUDED

#include <stddef.h>

/* A Chunk can be either free or in use. */
enum ChunkStatus {CHUNK_FREE, CHUNK_INUSE};

/* A Chunk is a sequence of Units.  The first Unit is a header that
   indicates the number of Units in the Chunk, whether the Chunk is
   free, and, if the Chunk is free, a pointer to the next Chunk in the
   free list. The last Unit is a footer that indicates the number of
   Units in the Chunk and, if the Chunk is free, a pointer to the
   previous Chunk in the free list. The Units between the header and
   footer are the payload. */

typedef struct Chunk *Chunk_T;

/*--------------------------------------------------------------------*/

/* The minimum number of units that a Chunk can contain. */

static const size_t MIN_UNITS_PER_CHUNK = 3;

/*--------------------------------------------------------------------*/

/* Translate uBytes, a number of bytes, to units. Return the result. */

size_t Chunk_bytesToUnits(size_t uBytes);

/*--------------------------------------------------------------------*/

/* Translate uUnits, a number of units, to bytes. Return the result. */

size_t Chunk_unitsToBytes(size_t uUnits);

/*--------------------------------------------------------------------*/

/* Return the address of the payload of oChunk. */

void *Chunk_toPayload(Chunk_T oChunk);

/*--------------------------------------------------------------------*/

/* Return the Chunk whose payload is pointed to by pv. */

Chunk_T Chunk_fromPayload(void *pv);

/*--------------------------------------------------------------------*/

/* Return the status of oChunk. */

enum ChunkStatus Chunk_getStatus(Chunk_T oChunk);

/*--------------------------------------------------------------------*/

/* Set the status of oChunk to eStatus. */

```

```c
void Chunk_setStatus(Chunk_T oChunk, enum ChunkStatus eStatus);

/*--------------------------------------------------------------------*/

/* Return oChunk's number of units. */

size_t Chunk_getUnits(Chunk_T oChunk);

/*--------------------------------------------------------------------*/

/* Set oChunk's number of units to uUnits. */

void Chunk_setUnits(Chunk_T oChunk, size_t uUnits);

/*--------------------------------------------------------------------*/

/* Return oChunk's next Chunk in the free list, or NULL if there
   is no next Chunk. */

Chunk_T Chunk_getNextInList(Chunk_T oChunk);

/*--------------------------------------------------------------------*/

/* Set oChunk's next Chunk in the free list to oNextChunk. */

void Chunk_setNextInList(Chunk_T oChunk, Chunk_T oNextChunk);

/*--------------------------------------------------------------------*/

/* Return oChunk's previous Chunk in the free list, or NULL if there
   is no previous Chunk. */

Chunk_T Chunk_getPrevInList(Chunk_T oChunk);

/*--------------------------------------------------------------------*/

/* Set oChunk's previous Chunk in the free list to oPrevChunk. */

void Chunk_setPrevInList(Chunk_T oChunk, Chunk_T oPrevChunk);

/*--------------------------------------------------------------------*/

/* Return oChunk's next Chunk in memory, or NULL if there is no
   next Chunk. Use oHeapEnd to determine if there is no next
   Chunk. oChunk's number of units must be set properly for this
   function to work. */

Chunk_T Chunk_getNextInMem(Chunk_T oChunk, Chunk_T oHeapEnd);

/*--------------------------------------------------------------------*/

/* Return oChunk's previous Chunk in memory, or NULL if there is no
   previous Chunk. Use oHeapStart to determine if there is no
   previous Chunk. The previous Chunk's number of units must be set
   properly for this function to work. */

Chunk_T Chunk_getPrevInMem(Chunk_T oChunk, Chunk_T oHeapStart);

/*--------------------------------------------------------------------*/

/* Return 1 (TRUE) if oChunk is valid, notably with respect to
   oHeapStart and oHeapEnd, or 0 (FALSE) otherwise. */

```

chunk4.h (Page 3 of 3)

```c
int Chunk_isValid(Chunk_T oChunk,
                    Chunk_T oHeapStart, Chunk_T oHeapEnd);

#endif
```

chunk4.c (Page 1 of 4)

```c
/*--------------------------------------------------------------------*/
/* chunk4.c                                                           */
/* Author: Bob Dondero                                                */
/*--------------------------------------------------------------------*/

#include "chunk4.h"
#include <stddef.h>
#include <stdio.h>
#include <assert.h>

/*--------------------------------------------------------------------*/

/* Physically a Chunk is a structure consisting of a number of units
   and an address. Logically a Chunk consists of multiple such
   structures. */

struct Chunk
{
   /* The number of units in the Chunk. The low-order bit
      stores the Chunk's status. */
   size_t uUnits;

   /* The address of an adjacent Chunk. */
   Chunk_T oAdjacentChunk;
};

/*--------------------------------------------------------------------*/

size_t Chunk_bytesToUnits(size_t uBytes)
{
   size_t uUnits;
   uUnits = ((uBytes - 1) / sizeof(struct Chunk)) + 1;
   uUnits++;  /* Allow room for a header. */
   uUnits++;  /* Allow room for a footer. */
   return uUnits;
}

/*--------------------------------------------------------------------*/

size_t Chunk_unitsToBytes(size_t uUnits)
{
   return uUnits * sizeof(struct Chunk);
}

/*--------------------------------------------------------------------*/

void *Chunk_toPayload(Chunk_T oChunk)
{
   assert(oChunk != NULL);

   return (void*)(oChunk + 1);
}

/*--------------------------------------------------------------------*/

Chunk_T Chunk_fromPayload(void *pv)
{
   assert(pv != NULL);

   return (Chunk_T)pv - 1;
}

/*--------------------------------------------------------------------*/
```

chunk4.c (Page 2 of 4)

```

enum ChunkStatus Chunk_getStatus(Chunk_T oChunk)
{
   assert(oChunk != NULL);

   return oChunk->uUnits & 1UL;
}

/*--------------------------------------------------------------------*/

void Chunk_setStatus(Chunk_T oChunk, enum ChunkStatus eStatus)
{
   assert(oChunk != NULL);
   assert((eStatus == CHUNK_FREE) || (eStatus == CHUNK_INUSE));

   oChunk->uUnits &= ~1UL;
   oChunk->uUnits |= eStatus;
}

/*--------------------------------------------------------------------*/

size_t Chunk_getUnits(Chunk_T oChunk)
{
   assert(oChunk != NULL);

   return oChunk->uUnits >> 1;
}

/*--------------------------------------------------------------------*/

void Chunk_setUnits(Chunk_T oChunk, size_t uUnits)
{
   assert(oChunk != NULL);
   assert(uUnits >= MIN_UNITS_PER_CHUNK);

   /* Set the Units in oChunk's header. */
   oChunk->uUnits &= 1UL;
   oChunk->uUnits |= uUnits << 1UL;

   /* Set the Units in oChunk's footer. */
   (oChunk + uUnits - 1)->uUnits = uUnits;
}

/*--------------------------------------------------------------------*/

Chunk_T Chunk_getNextInList(Chunk_T oChunk)
{
   assert(oChunk != NULL);

   return oChunk->oAdjacentChunk;
}

/*--------------------------------------------------------------------*/

void Chunk_setNextInList(Chunk_T oChunk, Chunk_T oNextChunk)
{
   assert(oChunk != NULL);

   oChunk->oAdjacentChunk = oNextChunk;
}

/*--------------------------------------------------------------------*/

```

chunk4.c (Page 3 of 4)

```
Chunk_T Chunk_getPrevInList(Chunk_T oChunk)
{
   assert(oChunk != NULL);

   return (oChunk + Chunk_getUnits(oChunk) - 1)->oAdjacentChunk;
}

/*--------------------------------------------------------------------*/

void Chunk_setPrevInList(Chunk_T oChunk, Chunk_T oPrevChunk)
{
   assert(oChunk != NULL);

   (oChunk + Chunk_getUnits(oChunk) - 1)->oAdjacentChunk = oPrevChunk;
}

/*--------------------------------------------------------------------*/

Chunk_T Chunk_getNextInMem(Chunk_T oChunk, Chunk_T oHeapEnd)
{
   Chunk_T oNextChunk;

   assert(oChunk != NULL);
   assert(oHeapEnd != NULL);
   assert(oChunk < oHeapEnd);

   oNextChunk = oChunk + Chunk_getUnits(oChunk);
   assert(oNextChunk <= oHeapEnd);

   if (oNextChunk == oHeapEnd)
      return NULL;
   return oNextChunk;
}

/*--------------------------------------------------------------------*/

Chunk_T Chunk_getPrevInMem(Chunk_T oChunk, Chunk_T oHeapStart)
{
   Chunk_T oPrevChunk;

   assert(oChunk != NULL);
   assert(oHeapStart != NULL);
   assert(oChunk >= oHeapStart);

   if (oChunk == oHeapStart)
      return NULL;

   oPrevChunk = oChunk - ((oChunk - 1)->uUnits);
   assert(oPrevChunk >= oHeapStart);

   return oPrevChunk;
}

/*--------------------------------------------------------------------*/

/* Return the number of units as stored in oChunk's footer. */

static size_t Chunk_getFooterUnits(Chunk_T oChunk)
{
   assert(oChunk != NULL);

   return (oChunk + Chunk_getUnits(oChunk) - 1)->uUnits;
}
```

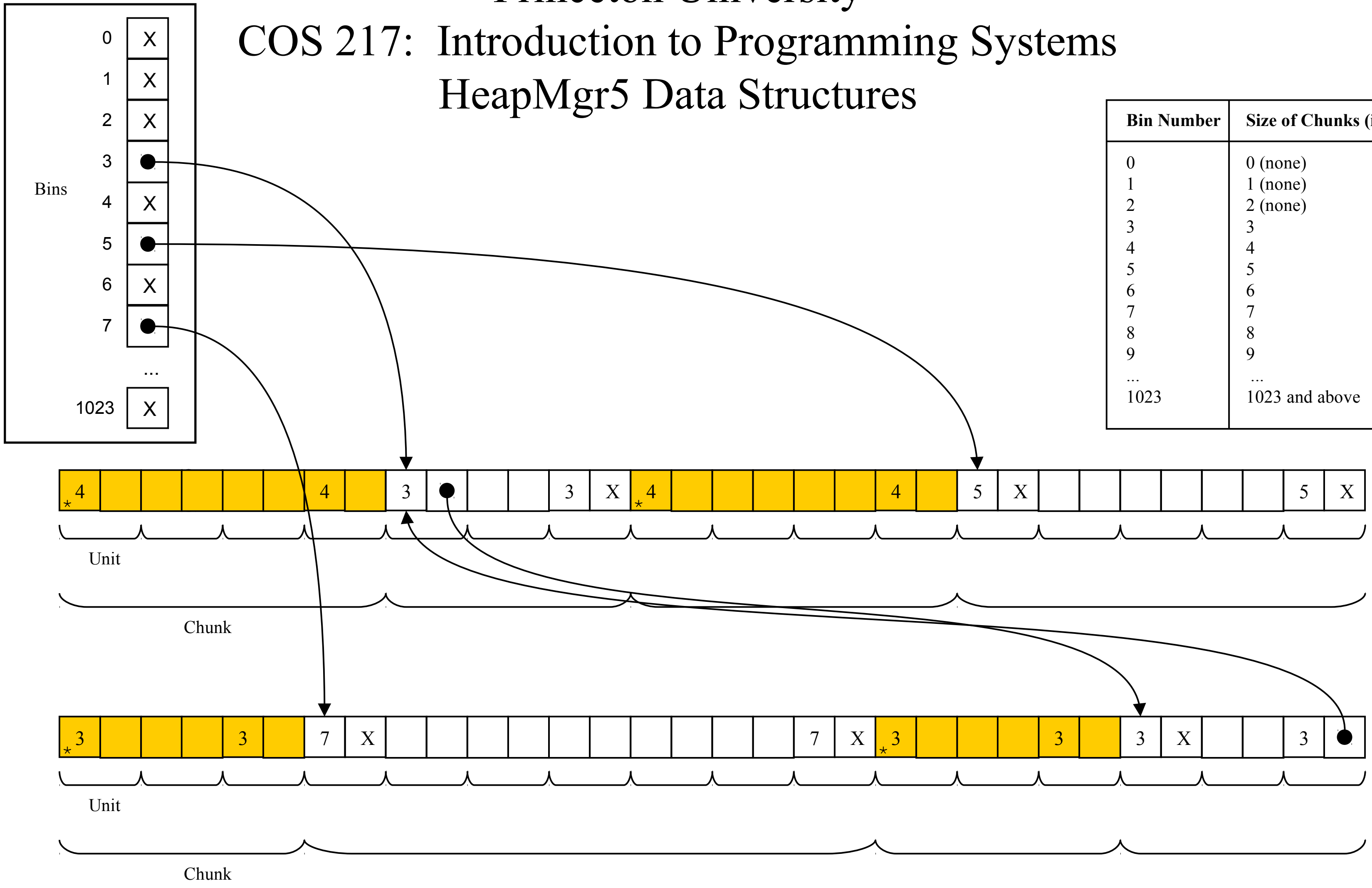
chunk4.c (Page 4 of 4)

```

/*--------------------------------------------------------------------*/

int Chunk_isValid(Chunk_T oChunk,
                  Chunk_T oHeapStart, Chunk_T oHeapEnd)
{
   assert(oChunk != NULL);
   assert(oHeapStart != NULL);
   assert(oHeapEnd != NULL);

   if (oChunk < oHeapStart)
   {  fprintf(stderr, "A chunk starts before the heap start\n");
      return 0;
   }
   if (oChunk >= oHeapEnd)
   {  fprintf(stderr, "A chunk starts after the heap end\n");
      return 0;
   }
   if (oChunk + Chunk_getUnits(oChunk) > oHeapEnd)
   {  fprintf(stderr, "A chunk ends after the heap end\n");
      return 0;
   }
   if (Chunk_getUnits(oChunk) == 0)
   {  fprintf(stderr, "A chunk has zero units\n");
      return 0;
   }
   if (Chunk_getUnits(oChunk) < MIN_UNITS_PER_CHUNK)
   {  fprintf(stderr, "A chunk has too few units\n");
      return 0;
   }
   if (Chunk_getUnits(oChunk) != Chunk_getFooterUnits(oChunk))
   {  fprintf(stderr, "A chunk has inconsistent header/footer sizes\n");
      return 0;
   }
   return 1;
}

```

checker4.h (Page 1 of 1)

```
/*--------------------------------------------------------------------*/
/* checker4.h                                                         */
/* Author: Bob Dondero                                                */
/*--------------------------------------------------------------------*/

#ifndef CHECKER4_INCLUDED
#define CHECKER4_INCLUDED

#include "chunk4.h"

/* Return 1 (TRUE) if the heap is in a valid state, or 0 (FALSE)
   otherwise. The heap is defined by parameters oHeapStart (the address
   of the start of the heap), oHeapEnd (the address immediately
   beyond the end of the heap), and oFreeList (a list containing free
   chunks). */

int Checker_isValid(Chunk_T oHeapStart, Chunk_T oHeapEnd,
   Chunk_T oFreeList);

#endif
```

Princeton University
COS 217: Introduction to Programming Systems
HeapMgr5 Data Structures



Bin Number	Size of Chunks (in Units)
0	0 (none)
1	1 (none)
2	2 (none)
3	3
4	4
5	5
6	6
7	7
8	8
9	9
...	...
1023	1023 and above

Each box consists of 8 bytes.
Each Chunk's header Unit contains a status (INUSE or FREE), a length, and, if the Chunk is free, a pointer to the next Chunk in its Bin.
Each Chunk's footer Unit contains a length and, if the Chunk is free, a pointer to the previous Chunk in its Bin.
The Chunks in the Bins are in no particular order.
* means INUSE; absence of * means FREE.

Copyright © 2018 by Robert M. Dondero, Jr.

Princeton University
COS 217: Introduction to Programming Systems
HeapMgr5 Algorithms

void *HeapMgr_malloc(size_t uBytes)

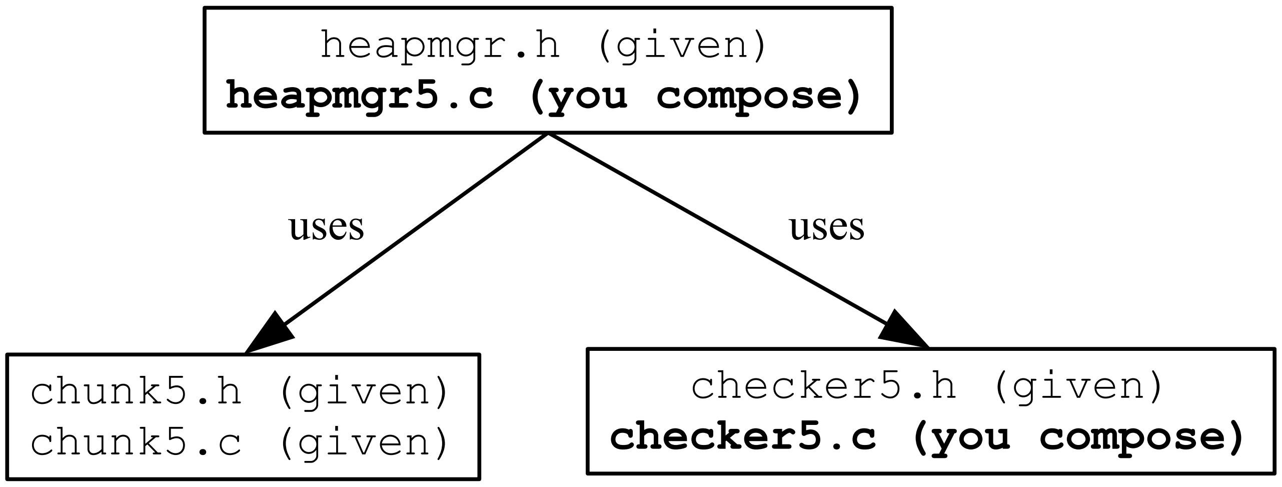
```
(1) If this is the first call of HeapMgr_malloc(), then initialize the heap manager.

(2) Determine the number of units the new chunk should contain.

(3) For each bin from the proper start bin to the last bin...

      For each chunk in the current bin...

         If the current chunk is big enough...

              If the current chunk is close to the requested size, then remove it from
              its bin, set its status to INUSE, and return it.  If the current chunk is
              too big, then remove it from its bin, split the chunk, insert the tail end
              of it into the proper bin at the front, set the status of the front end of
              it to INUSE, set the status of the tail end of it to FREE, and return the
              front end of it.

(4) Ask the OS for more memory — enough for the new chunk.  Return NULL if the OS refuses.
    Create a new chunk using that memory.  Insert the new chunk into the proper bin at the
    front.  If appropriate, coalesce the new chunk and the previous one in memory.  To do
    so, remove the current chunk from its bin, remove the previous chunk in memory from
    its bin, coalesce the two chunks to form a larger chunk, and insert the larger chunk
    into the proper bin at the front.  Let the current chunk be the new chunk.

(5) If the current chunk is close to the requested size, then remove it from its bin, set
    its status to INUSE, and return it.  If the current chunk is too big, then remove it
    from its bin, split the chunk, insert the tail end of it into the proper bin at the
    front, set the status of the front end of it to INUSE, set the status of the tail end
    of it to FREE, and return the front end of it.
```

void HeapMgr_free(void *pv)

```
(1) Set the status of the given chunk to FREE.

(2) Insert the given chunk into the proper bin at the front.

(3) If appropriate, coalesce the given chunk and the next one in memory.  To do so, remove
    the given chunk from its bin, remove the next chunk in memory from its bin, coalesce
    the two chunks to form a larger chunk, and insert the larger chunk into the proper bin
    at the front.

(4) If appropriate, coalesce the given chunk and the previous one in memory.  To do so,
    remove the given chunk from its bin, remove the previous chunk in memory from its bin,
    coalesce the two chunks to form a larger chunk, and insert the larger chunk into the
    proper bin at the front.
```

Copyright © 2018 by Robert M. Dondero, Jr.

Princeton University
COS 217: Introduction to Programming Systems
HeapMgr5 Code



Copyright © 2018 by Robert M. Dondero, Jr.

chunk5.h (Page 1 of 3)

```c
/*--------------------------------------------------------------------*/
/* chunk5.h                                                           */
/* Author: Bob Dondero                                                */
/*--------------------------------------------------------------------*/

#ifndef CHUNK5_INCLUDED
#define CHUNK5_INCLUDED

#include <stddef.h>

/* A Chunk can be either free or in use. */
enum ChunkStatus {CHUNK_FREE, CHUNK_INUSE};

/* A Chunk is a sequence of Units.  The first Unit is a header that
   indicates the number of Units in the Chunk, whether the Chunk is
   free, and, if the Chunk is free, a pointer to the next Chunk in the
   free list. The last Unit is a footer that indicates the number of
   Units in the Chunk and, if the Chunk is free, a pointer to the
   previous Chunk in the free list. The Units between the header and
   footer are the payload. */

typedef struct Chunk *Chunk_T;

/*--------------------------------------------------------------------*/

/* The minimum number of units that a Chunk can contain. */

static const size_t MIN_UNITS_PER_CHUNK = 3;

/*--------------------------------------------------------------------*/

/* Translate uBytes, a number of bytes, to units. Return the result. */

size_t Chunk_bytesToUnits(size_t uBytes);

/*--------------------------------------------------------------------*/

/* Translate uUnits, a number of units, to bytes. Return the result. */

size_t Chunk_unitsToBytes(size_t uUnits);

/*--------------------------------------------------------------------*/

/* Return the address of the payload of oChunk. */

void *Chunk_toPayload(Chunk_T oChunk);

/*--------------------------------------------------------------------*/

/* Return the Chunk whose payload is pointed to by pv. */

Chunk_T Chunk_fromPayload(void *pv);

/*--------------------------------------------------------------------*/

/* Return the status of oChunk. */

enum ChunkStatus Chunk_getStatus(Chunk_T oChunk);

/*--------------------------------------------------------------------*/

/* Set the status of oChunk to eStatus. */

```

```c
void Chunk_setStatus(Chunk_T oChunk, enum ChunkStatus eStatus);

/*--------------------------------------------------------------------*/

/* Return oChunk's number of units. */

size_t Chunk_getUnits(Chunk_T oChunk);

/*--------------------------------------------------------------------*/

/* Set oChunk's number of units to uUnits. */

void Chunk_setUnits(Chunk_T oChunk, size_t uUnits);

/*--------------------------------------------------------------------*/

/* Return oChunk's next Chunk in the free list, or NULL if there
   is no next Chunk. */

Chunk_T Chunk_getNextInList(Chunk_T oChunk);

/*--------------------------------------------------------------------*/

/* Set oChunk's next Chunk in the free list to oNextChunk. */

void Chunk_setNextInList(Chunk_T oChunk, Chunk_T oNextChunk);

/*--------------------------------------------------------------------*/

/* Return oChunk's previous Chunk in the free list, or NULL if there
   is no previous Chunk. */

Chunk_T Chunk_getPrevInList(Chunk_T oChunk);

/*--------------------------------------------------------------------*/

/* Set oChunk's previous Chunk in the free list to oPrevChunk. */

void Chunk_setPrevInList(Chunk_T oChunk, Chunk_T oPrevChunk);

/*--------------------------------------------------------------------*/

/* Return oChunk's next Chunk in memory, or NULL if there is no
   next Chunk. Use oHeapEnd to determine if there is no next
   Chunk. oChunk's number of units must be set properly for this
   function to work. */

Chunk_T Chunk_getNextInMem(Chunk_T oChunk, Chunk_T oHeapEnd);

/*--------------------------------------------------------------------*/

/* Return oChunk's previous Chunk in memory, or NULL if there is no
   previous Chunk. Use oHeapStart to determine if there is no
   previous Chunk. The previous Chunk's number of units must be set
   properly for this function to work. */

Chunk_T Chunk_getPrevInMem(Chunk_T oChunk, Chunk_T oHeapStart);

/*--------------------------------------------------------------------*/

/* Return 1 (TRUE) if oChunk is valid, notably with respect to
   oHeapStart and oHeapEnd, or 0 (FALSE) otherwise. */

```

chunk5.h (Page 3 of 3)

```
int Chunk_isValid(Chunk_T oChunk,
                  Chunk_T oHeapStart, Chunk_T oHeapEnd);

#endif
```

chunk5.c (Page 1 of 4)

```
/*--------------------------------------------------------------------*/
/* chunk5.c                                                           */
/* Author: Bob Dondero                                                */
/*--------------------------------------------------------------------*/

#include "chunk5.h"
#include <stddef.h>
#include <stdio.h>
#include <assert.h>

/*--------------------------------------------------------------------*/

/* Physically a Chunk is a structure consisting of a number of units
   and an address. Logically a Chunk consists of multiple such
   structures. */

struct Chunk
{
   /* The number of units in the Chunk. The low-order bit
      stores the Chunk's status. */
   size_t uUnits;

   /* The address of an adjacent Chunk. */
   Chunk_T oAdjacentChunk;
};

/*--------------------------------------------------------------------*/

size_t Chunk_bytesToUnits(size_t uBytes)
{
   size_t uUnits;
   uUnits = ((uBytes - 1) / sizeof(struct Chunk)) + 1;
   uUnits++;  /* Allow room for a header. */
   uUnits++;  /* Allow room for a footer. */
   return uUnits;
}

/*--------------------------------------------------------------------*/

size_t Chunk_unitsToBytes(size_t uUnits)
{
   return uUnits * sizeof(struct Chunk);
}

/*--------------------------------------------------------------------*/

void *Chunk_toPayload(Chunk_T oChunk)
{
   assert(oChunk != NULL);

   return (void*)(oChunk + 1);
}

/*--------------------------------------------------------------------*/

Chunk_T Chunk_fromPayload(void *pv)
{
   assert(pv != NULL);

   return (Chunk_T)pv - 1;
}

/*--------------------------------------------------------------------*/
```

chunk5.c (Page 2 of 4)

```

enum ChunkStatus Chunk_getStatus(Chunk_T oChunk)
{
   assert(oChunk != NULL);

   return oChunk->uUnits & 1UL;
}

/*--------------------------------------------------------------------*/

void Chunk_setStatus(Chunk_T oChunk, enum ChunkStatus eStatus)
{
   assert(oChunk != NULL);
   assert((eStatus == CHUNK_FREE) || (eStatus == CHUNK_INUSE));

   oChunk->uUnits &= ~1UL;
   oChunk->uUnits |= eStatus;
}

/*--------------------------------------------------------------------*/

size_t Chunk_getUnits(Chunk_T oChunk)
{
   assert(oChunk != NULL);

   return oChunk->uUnits >> 1;
}

/*--------------------------------------------------------------------*/

void Chunk_setUnits(Chunk_T oChunk, size_t uUnits)
{
   assert(oChunk != NULL);
   assert(uUnits >= MIN_UNITS_PER_CHUNK);

   /* Set the Units in oChunk's header. */
   oChunk->uUnits &= 1UL;
   oChunk->uUnits |= uUnits << 1UL;

   /* Set the Units in oChunk's footer. */
   (oChunk + uUnits - 1)->uUnits = uUnits;
}

/*--------------------------------------------------------------------*/

Chunk_T Chunk_getNextInList(Chunk_T oChunk)
{
   assert(oChunk != NULL);

   return oChunk->oAdjacentChunk;
}

/*--------------------------------------------------------------------*/

void Chunk_setNextInList(Chunk_T oChunk, Chunk_T oNextChunk)
{
   assert(oChunk != NULL);

   oChunk->oAdjacentChunk = oNextChunk;
}

/*--------------------------------------------------------------------*/

```

chunk5.c (Page 3 of 4)

```c
Chunk_T Chunk_getPrevInList(Chunk_T oChunk)
{
   assert(oChunk != NULL);

   return (oChunk + Chunk_getUnits(oChunk) - 1)->oAdjacentChunk;
}

/*--------------------------------------------------------------------*/

void Chunk_setPrevInList(Chunk_T oChunk, Chunk_T oPrevChunk)
{
   assert(oChunk != NULL);

   (oChunk + Chunk_getUnits(oChunk) - 1)->oAdjacentChunk = oPrevChunk;
}

/*--------------------------------------------------------------------*/

Chunk_T Chunk_getNextInMem(Chunk_T oChunk, Chunk_T oHeapEnd)
{
   Chunk_T oNextChunk;

   assert(oChunk != NULL);
   assert(oHeapEnd != NULL);
   assert(oChunk < oHeapEnd);

   oNextChunk = oChunk + Chunk_getUnits(oChunk);
   assert(oNextChunk <= oHeapEnd);

   if (oNextChunk == oHeapEnd)
      return NULL;
   return oNextChunk;
}

/*--------------------------------------------------------------------*/

Chunk_T Chunk_getPrevInMem(Chunk_T oChunk, Chunk_T oHeapStart)
{
   Chunk_T oPrevChunk;

   assert(oChunk != NULL);
   assert(oHeapStart != NULL);
   assert(oChunk >= oHeapStart);

   if (oChunk == oHeapStart)
      return NULL;

   oPrevChunk = oChunk - ((oChunk - 1)->uUnits);
   assert(oPrevChunk >= oHeapStart);

   return oPrevChunk;
}

/*--------------------------------------------------------------------*/

/* Return the number of units as stored in oChunk's footer. */

static size_t Chunk_getFooterUnits(Chunk_T oChunk)
{
   assert(oChunk != NULL);

   return (oChunk + Chunk_getUnits(oChunk) - 1)->uUnits;
}
```

chunk5.c (Page 4 of 4)

```
/*--------------------------------------------------------------------*/

int Chunk_isValid(Chunk_T oChunk,
                  Chunk_T oHeapStart, Chunk_T oHeapEnd)
{
   assert(oChunk != NULL);
   assert(oHeapStart != NULL);
   assert(oHeapEnd != NULL);

   if (oChunk < oHeapStart)
   {  fprintf(stderr, "A chunk starts before the heap start\n");
      return 0;
   }
   if (oChunk >= oHeapEnd)
   {  fprintf(stderr, "A chunk starts after the heap end\n");
      return 0;
   }
   if (oChunk + Chunk_getUnits(oChunk) > oHeapEnd)
   {  fprintf(stderr, "A chunk ends after the heap end\n");
      return 0;
   }
   if (Chunk_getUnits(oChunk) == 0)
   {  fprintf(stderr, "A chunk has zero units\n");
      return 0;
   }
   if (Chunk_getUnits(oChunk) < MIN_UNITS_PER_CHUNK)
   {  fprintf(stderr, "A chunk has too few units\n");
      return 0;
   }
   if (Chunk_getUnits(oChunk) != Chunk_getFooterUnits(oChunk))
   {  fprintf(stderr, "A chunk has inconsistent header/footer sizes\n");
      return 0;
   }
   return 1;
}

```

checker5.h (Page 1 of 1)

```c
/*--------------------------------------------------------------------*/
/* checker5.h                                                         */
/* Author: Bob Dondero                                                */
/*--------------------------------------------------------------------*/

#ifndef CHECKER5_INCLUDED
#define CHECKER5_INCLUDED

#include "chunk5.h"

/* Return 1 (TRUE) if the heap is in a valid state, or 0 (FALSE)
   otherwise. The heap is defined by parameters oHeapStart (the address
   of the start of the heap), oHeapEnd (the address immediately
   beyond the end of the heap), and aoBins (an array of iBinCount bins,
   where each bin contains free chunks). */

int Checker_isValid(Chunk_T oHeapStart, Chunk_T oHeapEnd,
   Chunk_T aoBins[], int iBinCount);

#endif
```