Koddokumentation

iMalloc.c

iMalloc
Funktionen anropas med storleken storleken på den virituella heap som användaren efterfrågar samt ett antal alternativ för funktioner som användaren specificerar. Beroende på vilka alternativ användaren har angett kommer koden att returnera en memorypekare av en viss storlek som innefattar ett antal funktionspekare motsvarande de alternativ som efterfrågades.

iMallocFree
Anropas med en memory pekare. Frigör sedan varje enskilt element i listan för att sedan frigöra det minne som iMalloc allokerat.

memoryAllocator.c

mkChunkFrom
Funktionen mkChunkFrom tar in en pekare till en chunk c samt en unsigned int bytes. Det som sedan händer och returneras är att det skapas en ny chunk newChunk som representerar det lediga utrymmet som blir över efter en split av c. För att funktionen ska fungera korrekt så krävs det att bytes är mindre än storleken på c.

descendingSort
Funktionen descendingSort tar in en lista list, samt två pekare till chunks, currentChunk och sortObj, och sorterar sedan in sortObj på rätt plats i chunklistan relativt till vad currentChunk pekar på. Den här funktionen sorterar chunksen i fallande storleksordning.
Vid behov uppdateras lists first- respektive last-pekare till att peka på den nyligen insorterade chunken.

ascendingSort
Funktionen ascendingSort tar in samma argument och arbetar på samma sätt som descendingSort men sorterar istället chunksen i växande storleksordning.

fits
Funktionen fits tar in en pekare till en chunk c, och en unsigned int bytes. Syftet är att undersöka om den mängd minne användaren vill allokera ryms någonstans i den heap som redan är skapad av iMalloc. Om det visar sig att bytes inte ryms i den aktuella chunken så stegar funktionen vidare i chunklistan. Om en chunk hittas som uppfyller detta villkor så returneras denna, annars returneras NULL.

splitAddress
Funktionen splitAddress tar in en pekare till en lista list, en pekare till en chunk c och en unsigned int bytes. När splitAddress anropas så jämförs antalet bytes som användaren vill allokera mot storleken på den aktuella chunken. Om storleken visas vara densamma så är det ej nödvändigt att förstöra den lediga chunken i mindre bitar och därför ändras bara aktuella värden (så som mark-bit).
Om storleken på användarens allokering är mindre än storleken på den aktuella chunken, och den är ledig, så delas chunken upp i två delar (en för det allokerade utrymmet och en för det återstående lediga utrymmet). Dessa chunks hamnar i addressordning.

splitAscDesc
Funktionen splitAscDesc tar in samma argument som splitAddress och fungerar på ungefär samma sätt som splitAddress, men är istället anpassad för storlekssortering av chunksen.

allocInt
Funktionen allocInt tar in en pekare till ett Memory mem och en unsigned int bytes. Den här funktionen anropas av användaren för att reservera utrymme för egna variabler och data på den via iMalloc skapade heapen och får sedan tillbaks en pekare till detta data. Då heapen är full eller funktionen anropas med bytes = 0 returneras en NULL-pekare, och ett felmeddelande om att det ej är möjligt att allokera 0 bytes i det senare fallet.

allocChar
Funktionen allocChar opererar på samma vis som allocInt med undantag för att istället för att ta in en unsigned int som andra argument tar in en formatsträng. Denna formatsträng kan innehålla vilka tecken som helst, men endast en begränsad uppsättning tecken (*, i, f, c, l, d) är av värde och översätts sedan till ett antal bytes. Formatsträngen ”i” översätts till exempel till storleken av en int.
Övriga tecken ignoreras.

availableMemory
Funktionen availableMemory tar in en Memory-pekare mem. Funktionen itererar över chunklistan (som räknas fram med hjälp av pekararitmetik från mem) och räknar ihop totalt ledigt minne i hela heapen som är allokerad av iMalloc.

memConcat
Funktion memConcat tar in en list-pekare list. Denna slår ihop lediga chunks som ligger bredvid varandra i minnet, och den nya storleken på den sammanslagna chunken blir summan av de två ursprungliga. För att funktionen ska fungera korrekt bör listans sortering vara inställd på adressortering.

freeMem
Funktionen freeMem tar in en Memory-pekare mem och en void-pekare object och användaren anropar denna funktion för att frigöra lagrad data på heapen (sätta mark-bit till 0).

collect.c

mark
Anropas ifrån traverseStack när den har hittat en pekare som pekar i den AddressSpace som traverseStack anropdes med.
Mark får in en pekare och en pChunk.
Allt den gör är att sätta chunkens mark till 1.

Här skulle vi vilja ha en jämförelse för att se om pekaren som funktionen anropades med ligger på den ”aktiva” delen av stacken eller inte. Om vi hade hunnit lägga till detta så skulle våran bugg vara löst.

collect
Anropas av användaren med användarens Memory struct (det som iMalloc returnerade).

Funktionen börjar med att skapa en pekare till listan som nås genom pekararitmetik. Sedan så anropar funktionen markZero för att räkna hur många fri chunks som finns i listan.
Därefter så kommer traverseStack anropas för samtliga chunks i listan, traverseStack får även med sig respektive AddressSpace för chunksens data och funktionen mark.

Om traveseStack markerat några chunks som aktiva så kommer traverseChunk anropas för de chunksen för att se om de aktiva chunksen innehåller några pekare till andra chunks.

Samtliga chunks som har pekare till sig har nu aktiverats.
Nu räknas de inaktiva chunksen och om de är fler nu än tidigare (det som markZero returnerade) så kommer funktionen returnera 1 annars 2.
Som sista steg innan returen så kollar vi om listan är sorterad efter addressplatser, om så är fallet så anropar vi memConcat.

markZero
Anropas ifrån collect med en pList.
Funktionen traverserar listan ifrån start till slut och ställer samtliga mark-bits till 0.
Den räknar hur många fria chunks listan innehåller och returnerar detta värde som en int.

mkAddressSpace
Anropas med två pekare som castast som RawPtr.
Skapar ett intervall mellan två addresser.
Returnerar adresspannet som en ”AddressSpace”.

traverseChunk
Anropas med en pChunk, ett AddressSpace samt en pLista.
Chunkens data traverseras bit för bit i sökan efter pekare.
När en pekare hittas så kontrolleras det ifall den pekar in i det AddressSpace som sickats med.
Om så är fallet så anropas findChunk.
Sedan fortsätter funktionen leta efter pekare.

findChunk
Funktionen anropas av traverseChunk med en pekare, samma pLista samt den pChunk som traverseChunk anropas med. Funktionen traverserar listan och letar efter den chunk vars data som innehåller minnesadressen som pekaren pekar på.
När korrekt chunk hittas så jämförs den med den medskickade chunken. Om de är samma så stannar funktionen. Om de ej är samma så anropas traverseChunk för den hittade chunken.

refcount.c

global variabel ”globalpList”
detta är en nödlösning för att lösa problemet med att hitta rätt chunk att modifiera refcount i. Vi har gjort så att vi allokerar minne utanför det minne som iMalloc självt allokerar för att spara meta data så att användaren får tillgång till allt det minne han har bett om. Så argumentet till de olika funktionerna i refcount blir en pekare till datat i chunken, inte informationen om chunken. För att då kunna hitta rätt chunk att ändra refcount i så måste vi ha adressen till det minne som vi allokerar vid anrop på alloc.

findAChunk
funktionen letar igenom listan (den globala variabeln) med minne som returneras vid anrop av alloc för att hitta rätt chunk som motsvarar argumentet till findAChunk funktionen.

chunkMem
Det denna funktion gör är att gå igenom en chunks void-datas samtliga minnesplatser för att se om det pekar in i någon annas chunks void-data. Den bryr sig inte om ifall den pekar in i någon annan adress som redan innefattas av den ursprungliga chunkens void-data. Om det skulle vara så att vi hittar en sådan pekare så skall funktionen release anropas på den chunken.

retain
går in i önskad chunk efter att ha letat rätt på den med hjälp av findAChunk funktionen och ökar på dess refcount med ett. Den returnerar sedan det talet.

count
går in i önskad chunk efter att ha letat rätt på den med hjälp av findAChunk funktionen och returnerar sedan det talet.

release
funktionen minskar refcount talet med ett. Om det visar sig att refcount blir noll så skall den chunken frias, den har ju då inga pekare till sig och går således inte att komma åt. Vid det tillfället så går funktionen även igenom alla interna minnesplatser i void-datat som chunken pekar på för att kontrollera att ingen av dem pekar på någon annan chunks void-data. I så fall måste även dess refcount minska med ett eftersom vi tar bort en av dess pekare när vi friar chunken.