1 Considerazioni generali

Le uniche strutture dati che ho usato per gestire la memoria interna di storman fino alla parte C sono state liste. Ho utilizzato per la precisione

- Una lista per gestire le zone. All'interno di ogni nodo ho messo le informazioni:
  - *placement*: ovvero l'indirizzo di memoria di inizio della zona;
  - *tot_mem*: ovvero la memoria totale allocata nella zona;
  - *block_listhead*: ovvero il puntatore alla testa della lista dei blocchi della zona;
- Una lista ordinata (secondo il valore di *placement*) per gestire i blocchi di ogni zona. All'interno di ogni nodo ho messo le informazioni:
  - *placement*: ovvero l'indirizzo di memoria di inizio del blocco;
  - *tot_mem*: ovvero la memoria totale allocata nel blocco;
  - *alignment*: ovvero l'allineamento richiesto dall'utente in block_alloc. Per i blocchi liberi ha valore 1;
  - *block_listhead*: ovvero il puntatore alla testa della lista dei puntatori al blocco. Se questo puntatore è NULL vuol dire che non ci sono puntatori, ovvero che il blocco è "libero";

  Una lista per gestire i puntatori al blocco. All'interno di ogni nodo ho messo le informazioni:
  - *ptr*: ovvero l'indirizzo di memoria a cui punta il puntatore all'interno del blocco;
  - *ptrptr*: ovvero il puntatore all'indirizzo di memoria fornito con *block_alloc* o altre funzioni per assegnare puntatori dall'utente.

2 Libreria *funzionivarie*

In questa libreria sono presenti le funzioni di base e di utilizzo generale per *storman*. Oltre a funzioni di inserimento e di eliminazione di nodi all'interno delle liste abbiamo:

- *zone_alloc*: Una funzione che alloca con *malloc* una zona di dimensione doppia rispetto al *size* fornito dall'utente;

- *clean_ zone*: serve per unire blocchi liberi della zona consecutivi in un unico blocco;
- *lookfor*: cerca all'interno dei puntatori di *storman* se c'è un puntatore con *ptrptr* uguale a un valore fornito. Se sì ritorna 1, altrimenti 0. Passando per indirizzo dei puntatori alle *struct zone*, *block* e *list_ blockPtr* assegna a questi valori la zona, il blocco e il puntatore in cui è stato trovato il valore.
- *lookfor_ block*: Cerca all'interno di storman se il valore passato è un indirizzo di memoria di un blocco di *storman*. Se sì ritorna 1, 0 altrimenti. Similmente alla funzione precendente, passando per indirizzo dei puntatori alle *struct zone* e *block*, assegna a questi valori la zona e il blocco in cui è stato trovato il valore.
- *count_ storman_ block*: Funzione che serve a contare il numero di blocchi occupati di storman

3 Parte A

Analizzerò più da vicino il funzionamento delle prime due funzioni della parte A. *pointer_ assign* e *pointer_ release* sono meno interessanti e sono ampiamente spiegate nei commenti al codice.

3.1 *block_ alloc*

Il funzionamento è divisibile in tre parti. In primis controlla che *alignment* sia accettabile con la funzione ricorsiva *isAccetable*. Poi controlla se l'indirizzo in *ptr_ addr* è già gestito con la funzione *lookfor* e se sì applico *block_ release*. Da qui in poi abbiamo la certezza che l'indirizzo in *ptr_ addr* non è già gestito da *storman*, cerco un blocco libero che abbia abbastanza memoria, facendo attenziona anche allo spazio vuoto da lasciare a causa dell'allineamento. Se non lo trovo allora applico *zone_ alloc* e rifaccio la ricerca che avrà adesso sicuramente successo. Trovato il blocco, aggiorno le mie strutture dati interne e ritorno i valori richiesti.

3.2 *block_ release*

Cerca con *lookfor* il puntatore di *storman* da rilasciare e lo rilascia. Controlla se il blocco adesso è diventatato libero, se sì bisogna quindi solo ripulire la zona con *clean_ zone*. Controlla infine se nella zona è rimasto un solo blocco

e se questo blocco è libero, questo significa che non c'è più nessun puntatore a quella zona, dunque la rilascio.

4 Parte B

La funzione *block_ info* segue quasi letteralmente le indicazioni fornite ed è ampiamente spiegata nei commenti. La *macro assign* strutta le funzioni *lookfor* e *lookfor_ block* per per verificare le condizioni 1 e 2, e assegna a *lv* il valore *rv* se non sono verificate.

4.1 *pointer_ info*

Il codice è molto breve. Per prima cosa controlla che *ptr_ addr* sia effettivamente gestito da *storman*. Cerca poi se *ptr_ addr* è all'interno di un blocco di *storman*, in tal caso lo storage del puntatore con indirizzo in *ptr_ addr* è di tipo dinamico, altrimenti è di tipo statico o automatico.

4.2 *block_ realloc*

block_ realloc ha un funzionamento molto diverso a seconda del valore di *newsize*. Abbiamo tre casi:

1. se $|B| = newsize$ allora non c'è nulla da fare
2. se $newsize < |B|$ allora:
   - Controlla con la funzione *ptrMoreThanSize* se ci sono dei puntatori che non permettono il restringimento del blocco, se ci sono ritorna 2;
   - Se posso restringere il blocco allora aggiorna i valori di *tot_ mem* del blocco. Adesso abbiamo due casi: se c'è un blocco successivo all'interno della zona e questo blocco è libero, aggiorno l'indirizzo di inizio del blocco libero successivo e la sua *tot_ mem*; se non c'è un blocco successivo o è occupato allora creo un nuovo blocco libero che riempie lo spazio vuoto rimasto nel restringere il blocco.
3. Se $newsize > |B|$ allora:
   - Controllo se ho spazio a destra del blocco con la funzione *thereIsRightSpace*;

- Se ho trovato dello spazio, aggiorno il valore *tot_mem* del mio blocco e restringo o cancello fino ai 2 blocchi successivi (il caso peggiore è se devo cancellare il blocco libero successivo e restringere il blocco occupato ancora dopo);
- Se non ho trovato dello spazio libero allora: alloco un nuovo blocco con $tot\_mem = newsize$, copio con la funzione *myMemCopy* byte per byte lo storage nel nuovo blocco, sposto i puntatori del vecchio blocco nel nuovo blocco con *pointer_assign* ed infine dealloco il vecchio blocco con *block_release*.

5 Parte C

5.1 Libreria *funzioniParteC*

Per la parte C è stato necessario creare nuove strutture dati:

- Una coda di struct *list_blockPtr*;
- Un grafo orientato di blocchi dove ogni blocco è collegato a quelli raggiungibili (nella definizione data nella spiegazione di *sweeping_release* nel pdf di presentazione del progetto) in un solo passo.
- Una lista di interi che mi servirà per fare liste di elementi del grafo (indentificati dagli indici del vettore in cui è salvato il grafo).

In questa libreria abbiamo le funzioni di base per queste strutture. Sono presenti inoltre le funzioni:

- *block_enqueue*: che inserisce nella coda i puntatori il cui storage è contenuto nel blocco fornito;
- *create_block_graph*: che crea il grafo;
- *visita*: che è una visita in profondità del grafo.

5.2 *ext_block_release*

ext_block_release in primis inserisce nella coda il puntatore di *storman* corrispondente a *ptr_addr*. Poi itera questo procedimento fino all'esaurimento della coda: estrae un elemento della coda, cerca il blocco corrispondente e controlla se ha un solo puntatore e se sì inserisce nella coda tutti i puntatori il cui storage è contenuto in B con *block_enqueue*, infine rilascia il puntatore ed eventualmente il blocco corrispondente con *block_release*.

5.3 *ext_ block_ alloc*

Controlla se *ptr_ addr* è già gestito da *storman*. Se sì applica *ext_ block_ release* e poi *block_ alloc*, altrimenti applica solo *block_ alloc*.

5.4 sweeping_release

Ho implementato *sweeping_ release* come la ricerca delle componenti connesse di un grafo. Creando un grafo come spiegato in precedenza e facendo una visita a partire da ogni blocco che ha in sè un puntatore con storage statico o automatico marco come visitati tutti i blocchi che non devo eliminare. Tutti gli altri blocchi li rilascio iterando *block_ release*.