Architettura dei Calcolatori – Laboratorio 2: StarWars Realloc – Esercizio 1

Gruppo:
Infusini Stefano 5211120
Di Luca Mattia 5213177
Trioli Davide 5316731

Il programma *malloc_play* utilizza funzioni *malloc*, *realloc*, *free* per osservare i loro comportamenti in memoria.

Nella prima parte il programma alloca in memoria una dimensione pari a 1 byte denominandola “p” attraverso la funzione *malloc*:

```
p = (unsigned char*)malloc(sz);
```

e inserendo la corrispondente lettera 'p' nella posizione appena allocata.

Osservando il comportamento della memoria prima e dopo la posizione allocata si può notare questo comportamento:

```
... allocating 1 unsigned chars to pointer p, min=-16, max=60

p[-16]=  0      p[-15]=  0      p[-14]=  0      p[-13]=  0
p[-12]=  0      p[-11]=  0      p[-10]=  0      p[ -9]=  0
p[ -8]= 33      p[ -7]=  0      p[ -6]=  0      p[ -5]=  0
p[ -4]=  0      p[ -3]=  0      p[ -2]=  0      p[ -1]=  0
p[  0]=112      p[  1]=  0      p[  2]=  0      p[  3]=  0
p[  4]=  0      p[  5]=  0      p[  6]=  0      p[  7]=  0
p[  8]=  0      p[  9]=  0      p[ 10]=  0      p[ 11]=  0
p[ 12]=  0      p[ 13]=  0      p[ 14]=  0      p[ 15]=  0
p[ 16]=  0      p[ 17]=  0      p[ 18]=  0      p[ 19]=  0
p[ 20]=  0      p[ 21]=  0      p[ 22]=  0      p[ 23]=  0
p[ 24]= 65      p[ 25]=253      p[ 26]=  1      p[ 27]=  0
p[ 28]=  0      p[ 29]=  0      p[ 30]=  0      p[ 31]=  0
p[ 32]=  0      p[ 33]=  0      p[ 34]=  0      p[ 35]=  0
p[ 36]=  0      p[ 37]=  0      p[ 38]=  0      p[ 39]=  0
p[ 40]=  0      p[ 41]=  0      p[ 42]=  0      p[ 43]=  0
p[ 44]=  0      p[ 45]=  0      p[ 46]=  0      p[ 47]=  0
p[ 48]=  0      p[ 49]=  0      p[ 50]=  0      p[ 51]=  0
p[ 52]=  0      p[ 53]=  0      p[ 54]=  0      p[ 55]=  0
p[ 56]=  0      p[ 57]=  0      p[ 58]=  0      p[ 59]=  0
p[ 60]=  0
```

In posizione p[0] si può notare come sia finita effettivamente la lettera 'p', denotata dal suo codice ASCII 112.
Nelle posizioni circostanti si possono notare però dei numeri particolari, differenti da memoria “random” che ci si potrebbe aspettare da zone non allocate. Questo comportamento deriva dal funzionamento di *malloc*, ad esempio:
8 byte prima della zona allocata si trova un 33;
24 byte dopo la posizione allocata si trovano, in ordine, 65, 253 e 1.
Questo comportamento si può spiegare proprio grazie all'utilizzo di *malloc*, che alloca una dimensione in memoria a blocchi più grande di quella strettamente richiesta (1 byte) in modo da averla “pronta” in caso il programmatore decidesse di averne bisogno (magari attraverso una *realloc*, come si può vedere successivamente) e delimita i confini della memoria allocata con dei dati relativi, che verranno esplicitati più avanti.

Allocando successivamente una zona di memoria denominata “q” e osservando il suo intorno come prima, si può vedere come questa venga allocata dopo quella precedente “p”:

```
... allocating 1 more unsigned chars to a different pointer q

q[-48]=  0      q[-47]=  0      q[-46]=  0      q[-45]=  0
q[-44]=  0      q[-43]=  0      q[-42]=  0      q[-41]=  0
q[-40]= 33      q[-39]=  0      q[-38]=  0      q[-37]=  0
q[-36]=  0      q[-35]=  0      q[-34]=  0      q[-33]=  0
q[-32]=112      q[-31]=  0      q[-30]=  0      q[-29]=  0
q[-28]=  0      q[-27]=  0      q[-26]=  0      q[-25]=  0
q[-24]=  0      q[-23]=  0      q[-22]=  0      q[-21]=  0
q[-20]=  0      q[-19]=  0      q[-18]=  0      q[-17]=  0
q[-16]=  0      q[-15]=  0      q[-14]=  0      q[-13]=  0
q[-12]=  0      q[-11]=  0      q[-10]=  0      q[ -9]=  0
q[ -8]= 33      q[ -7]=  0      q[ -6]=  0      q[ -5]=  0
q[ -4]=  0      q[ -3]=  0      q[ -2]=  0      q[ -1]=  0
q[  0]=113      q[  1]=  0      q[  2]=  0      q[  3]=  0
q[  4]=  0      q[  5]=  0      q[  6]=  0      q[  7]=  0
q[  8]=  0      q[  9]=  0      q[ 10]=  0      q[ 11]=  0
q[ 12]=  0      q[ 13]=  0      q[ 14]=  0      q[ 15]=  0
q[ 16]=  0      q[ 17]=  0      q[ 18]=  0      q[ 19]=  0
q[ 20]=  0      q[ 21]=  0      q[ 22]=  0      q[ 23]=  0
q[ 24]= 33      q[ 25]=253      q[ 26]=  1      q[ 27]=  0
q[ 28]=  0      q[ 29]=  0      q[ 30]=  0      q[ 31]=  0
q[ 32]=  0      q[ 33]=  0      q[ 34]=  0      q[ 35]=  0
q[ 36]=  0      q[ 37]=  0      q[ 38]=  0      q[ 39]=  0
q[ 40]=  0      q[ 41]=  0      q[ 42]=  0      q[ 43]=  0
q[ 44]=  0      q[ 45]=  0      q[ 46]=  0      q[ 47]=  0
q[ 48]=  0      q[ 49]=  0      q[ 50]=  0      q[ 51]=  0
q[ 52]=  0      q[ 53]=  0      q[ 54]=  0      q[ 55]=  0
q[ 56]=  0      q[ 57]=  0      q[ 58]=  0      q[ 59]=  0
q[ 60]=  0
```

Si può notare come esista sempre un valore 33 posto 8 byte prima della zona di memoria; questa volta però nel primo dei 3 byte “in fondo” non figura più un 65 ma un 33 - una differenza di 32, che si può ricollegare alla differenza di posizionamento in memoria delle due zone allocate: ciò significa che questo valore sta a significare quanta memoria è stata consumata all'interno di un blocco, avendo esattamente 32 byte in meno disponibili.

Successiva operazione effettuata è di *realloc* della zona “p”, questa volta di 5 byte al posto di 1 e inserendo una stringa di valore “startp”. Inoltre viene indirizzato un puntatore “oldp” alla precedente posizione di “p”:

```
oldp = p;
p = (unsigned char*)realloc((void*)p,sz);
check_pointer(p)
sprintf(p,"startp");
```

Da come si può notare:

```
... reallocating 5 unsigned chars to p

p[-16]=  0      p[-15]=  0      p[-14]=  0      p[-13]=  0
p[-12]=  0      p[-11]=  0      p[-10]=  0      p[ -9]=  0
p[ -8]= 33      p[ -7]=  0      p[ -6]=  0      p[ -5]=  0
p[ -4]=  0      p[ -3]=  0      p[ -2]=  0      p[ -1]=  0
p[  0]=115      p[  1]=116      p[  2]= 97      p[  3]=114
p[  4]=116      p[  5]=112      p[  6]=  0      p[  7]=  0
p[  8]=  0      p[  9]=  0      p[ 10]=  0      p[ 11]=  0
p[ 12]=  0      p[ 13]=  0      p[ 14]=  0      p[ 15]=  0
p[ 16]=  0      p[ 17]=  0      p[ 18]=  0      p[ 19]=  0
p[ 20]=  0      p[ 21]=  0      p[ 22]=  0      p[ 23]=  0
p[ 24]= 33      p[ 25]=  0      p[ 26]=  0      p[ 27]=  0
p[ 28]=  0      p[ 29]=  0      p[ 30]=  0      p[ 31]=  0
p[ 32]=113      p[ 33]=  0      p[ 34]=  0      p[ 35]=  0
p[ 36]=  0      p[ 37]=  0      p[ 38]=  0      p[ 39]=  0
p[ 40]=  0      p[ 41]=  0      p[ 42]=  0      p[ 43]=  0
p[ 44]=  0      p[ 45]=  0      p[ 46]=  0      p[ 47]=  0
p[ 48]=  0      p[ 49]=  0      p[ 50]=  0      p[ 51]=  0
p[ 52]=  0      p[ 53]=  0      p[ 54]=  0      p[ 55]=  0
p[ 56]= 33      p[ 57]=253      p[ 58]=  1      p[ 59]=  0
p[ 60]=  0
```

a partire dalla posizione p[0] viene correttamente inserita la stringa “startp” (qui presenti i valori ASCII dei caratteri), mentre successivamente nella posizione p[32] si può notare la lettera “q” inserita precedentemente e non modificata: questo perché la prima *malloc* effettuata, allocando preventivamente una dimensione di 24 byte accetta un ridimensionamento a 5 (essendo 5 < 24) senza dover liberare questa zona e riallocarne una nuova differente. Notare anche come gli ultimi byte segnino sempre 33, oltre che 253 e 1: la memoria allocata è di fatto rimasta la stessa anche dopo questa piccola *realloc*.
Questo è osservabile anche attraverso la stampa di “oldp”, che infatti punta esattamente alla stessa area di memoria di “p”:

```
 97 oldp[-16]=  0   oldp[-15]=  0   oldp[-14]=  0   oldp[-13]=  0
 98 oldp[-12]=  0   oldp[-11]=  0   oldp[-10]=  0   oldp[ -9]=  0
 99 oldp[ -8]= 33   oldp[ -7]=  0   oldp[ -6]=  0   oldp[ -5]=  0
100 oldp[ -4]=  0   oldp[ -3]=  0   oldp[ -2]=  0   oldp[ -1]=  0
101 oldp[  0]=115   oldp[  1]=116   oldp[  2]= 97   oldp[  3]=114
102 oldp[  4]=116   oldp[  5]=112   oldp[  6]=  0   oldp[  7]=  0
103 oldp[  8]=  0   oldp[  9]=  0   oldp[ 10]=  0   oldp[ 11]=  0
104 oldp[ 12]=  0   oldp[ 13]=  0   oldp[ 14]=  0   oldp[ 15]=  0
105 oldp[ 16]=  0   oldp[ 17]=  0   oldp[ 18]=  0   oldp[ 19]=  0
106 oldp[ 20]=  0   oldp[ 21]=  0   oldp[ 22]=  0   oldp[ 23]=  0
107 oldp[ 24]= 33   oldp[ 25]=  0   oldp[ 26]=  0   oldp[ 27]=  0
108 oldp[ 28]=  0   oldp[ 29]=  0   oldp[ 30]=  0   oldp[ 31]=  0
109 oldp[ 32]=113   oldp[ 33]=  0   oldp[ 34]=  0   oldp[ 35]=  0
110 oldp[ 36]=  0   oldp[ 37]=  0   oldp[ 38]=  0   oldp[ 39]=  0
111 oldp[ 40]=  0   oldp[ 41]=  0   oldp[ 42]=  0   oldp[ 43]=  0
112 oldp[ 44]=  0   oldp[ 45]=  0   oldp[ 46]=  0   oldp[ 47]=  0
113 oldp[ 48]=  0   oldp[ 49]=  0   oldp[ 50]=  0   oldp[ 51]=  0
114 oldp[ 52]=  0   oldp[ 53]=  0   oldp[ 54]=  0   oldp[ 55]=  0
115 oldp[ 56]= 33   oldp[ 57]=253   oldp[ 58]=  1   oldp[ 59]=  0
116 oldp[ 60]=  0
```

La successiva operazione prevede una riallocazione di “p”, questa volta di 40 byte rispetto ai 5 già allocati, inserendo anche una stringa di valore “modifiedp” che in effetti appare:

```
... reallocating 40 unsigned chars to p

p[-16]=  0      p[-15]=  0      p[-14]=  0      p[-13]=  0
p[-12]=  0      p[-11]=  0      p[-10]=  0      p[ -9]=  0
p[ -8]= 49      p[ -7]=  0      p[ -6]=  0      p[ -5]=  0
p[ -4]=  0      p[ -3]=  0      p[ -2]=  0      p[ -1]=  0
p[  0]=109      p[  1]=111      p[  2]=100      p[  3]=105
p[  4]=102      p[  5]=105      p[  6]=101      p[  7]=100
p[  8]=112      p[  9]=  0      p[ 10]=  0      p[ 11]=  0
p[ 12]=  0      p[ 13]=  0      p[ 14]=  0      p[ 15]=  0
p[ 16]=  0      p[ 17]=  0      p[ 18]=  0      p[ 19]=  0
p[ 20]=  0      p[ 21]=  0      p[ 22]=  0      p[ 23]=  0
p[ 24]=  0      p[ 25]=  0      p[ 26]=  0      p[ 27]=  0
p[ 28]=  0      p[ 29]=  0      p[ 30]=  0      p[ 31]=  0
p[ 32]=  0      p[ 33]=  0      p[ 34]=  0      p[ 35]=  0
p[ 36]=  0      p[ 37]=  0      p[ 38]=  0      p[ 39]=  0
p[ 40]=241      p[ 41]=252      p[ 42]=  1      p[ 43]=  0
p[ 44]=  0      p[ 45]=  0      p[ 46]=  0      p[ 47]=  0
p[ 48]=  0      p[ 49]=  0      p[ 50]=  0      p[ 51]=  0
p[ 52]=  0      p[ 53]=  0      p[ 54]=  0      p[ 55]=  0
p[ 56]=  0      p[ 57]=  0      p[ 58]=  0      p[ 59]=  0
p[ 60]=  0
```

Da qui si può notare come la funzione *realloc* questa volta abbia avuto la necessità di allocare una nuova area di memoria, in quanto la precedente prevedeva solo un massimo di 24 byte: infatti, non si nota più la presenza della lettera “q” in posizione p[32], e inoltre i numeri precedenti e successivi sono modificati: a p[-8] si nota un 49, numero di byte allocati in totale (8 prima e 40 dopo la posizione puntata da “p”, oltre che “p” stesso), e in fondo si notano 241, 252, 1. Questo significa che *realloc* ha occupato un nuovo “blocco”, diminuendo il secondo indice da 253 a 252; questo nuovo blocco, essendo la sua occupazione appena iniziata, presenta un 241 come primo indice.

Osservando quindi anche il “vecchio” puntatore “oldp”:

```
oldp[-16]=  0   oldp[-15]=  0   oldp[-14]=  0   oldp[-13]=  0
oldp[-12]=  0   oldp[-11]=  0   oldp[-10]=  0   oldp[ -9]=  0
oldp[ -8]= 33   oldp[ -7]=  0   oldp[ -6]=  0   oldp[ -5]=  0
oldp[ -4]=  0   oldp[ -3]=  0   oldp[ -2]=  0   oldp[ -1]=  0
oldp[  0]=105   oldp[  1]= 43   oldp[  2]=204   oldp[  3]= 89
oldp[  4]=  5   oldp[  5]=  0   oldp[  6]=  0   oldp[  7]=  0
oldp[  8]= 16   oldp[  9]=128   oldp[ 10]=182   oldp[ 11]=194
oldp[ 12]=156   oldp[ 13]= 85   oldp[ 14]=  0   oldp[ 15]=  0
oldp[ 16]=  0   oldp[ 17]=  0   oldp[ 18]=  0   oldp[ 19]=  0
oldp[ 20]=  0   oldp[ 21]=  0   oldp[ 22]=  0   oldp[ 23]=  0
oldp[ 24]= 33   oldp[ 25]=  0   oldp[ 26]=  0   oldp[ 27]=  0
oldp[ 28]=  0   oldp[ 29]=  0   oldp[ 30]=  0   oldp[ 31]=  0
oldp[ 32]=113   oldp[ 33]=  0   oldp[ 34]=  0   oldp[ 35]=  0
oldp[ 36]=  0   oldp[ 37]=  0   oldp[ 38]=  0   oldp[ 39]=  0
oldp[ 40]=  0   oldp[ 41]=  0   oldp[ 42]=  0   oldp[ 43]=  0
oldp[ 44]=  0   oldp[ 45]=  0   oldp[ 46]=  0   oldp[ 47]=  0
oldp[ 48]=  0   oldp[ 49]=  0   oldp[ 50]=  0   oldp[ 51]=  0
oldp[ 52]=  0   oldp[ 53]=  0   oldp[ 54]=  0   oldp[ 55]=  0
oldp[ 56]= 49   oldp[ 57]=  0   oldp[ 58]=  0   oldp[ 59]=  0
oldp[ 60]=  0
```

si può vedere come prima della posizione 32 corrispondente alla “q” questa volta vi siano dei valori “spazzatura”, valori che denotano una deallocazione di questo spazio di memoria in favore di quello nuovo precedentemente descritto.

Una successiva operazione di *realloc* su "p", questa volta diminuito da 40 a 5 byte, denota come il vecchio indirizzo, il primo, non venga più utilizzato ma venga comunque mantenuto l'indirizzo nuovo.

```
... reallocating 5 unsigned chars to p (again)

p[-16]=  0      p[-15]=  0      p[-14]=  0      p[-13]=  0
p[-12]=  0      p[-11]=  0      p[-10]=  0      p[ -9]=  0
p[ -8]= 49      p[ -7]=  0      p[ -6]=  0      p[ -5]=  0
p[ -4]=  0      p[ -3]=  0      p[ -2]=  0      p[ -1]=  0
p[  0]=109      p[  1]=111      p[  2]=100      p[  3]=105
p[  4]=102      p[  5]=105      p[  6]=101      p[  7]=100
p[  8]=112      p[  9]=  0      p[ 10]=  0      p[ 11]=  0
p[ 12]=  0      p[ 13]=  0      p[ 14]=  0      p[ 15]=  0
p[ 16]=  0      p[ 17]=  0      p[ 18]=  0      p[ 19]=  0
p[ 20]=  0      p[ 21]=  0      p[ 22]=  0      p[ 23]=  0
p[ 24]=  0      p[ 25]=  0      p[ 26]=  0      p[ 27]=  0
p[ 28]=  0      p[ 29]=  0      p[ 30]=  0      p[ 31]=  0
p[ 32]=  0      p[ 33]=  0      p[ 34]=  0      p[ 35]=  0
p[ 36]=  0      p[ 37]=  0      p[ 38]=  0      p[ 39]=  0
p[ 40]=241      p[ 41]=252      p[ 42]=  1      p[ 43]=  0
p[ 44]=  0      p[ 45]=  0      p[ 46]=  0      p[ 47]=  0
p[ 48]=  0      p[ 49]=  0      p[ 50]=  0      p[ 51]=  0
p[ 52]=  0      p[ 53]=  0      p[ 54]=  0      p[ 55]=  0
p[ 56]=  0      p[ 57]=  0      p[ 58]=  0      p[ 59]=  0
p[ 60]=  0
```

L'ultima operazione di *free* sul puntatore "p" (e successivamente su "q", in modo analogo) denota ancora una volta come durante questa fase la memoria venga riempita di valori che indicano una memoria libera non utilizzata.

```
... freeing p

p[-16]=  0      p[-15]=  0      p[-14]=  0      p[-13]=  0
p[-12]=  0      p[-11]=  0      p[-10]=  0      p[ -9]=  0
p[ -8]= 49      p[ -7]=  0      p[ -6]=  0      p[ -5]=  0
p[ -4]=  0      p[ -3]=  0      p[ -2]=  0      p[ -1]=  0
p[  0]=105      p[  1]= 43      p[  2]=204      p[  3]= 89
p[  4]=  5      p[  5]=  0      p[  6]=  0      p[  7]=  0
p[  8]= 16      p[  9]=128      p[ 10]=182      p[ 11]=194
p[ 12]=156      p[ 13]= 85      p[ 14]=  0      p[ 15]=  0
p[ 16]=  0      p[ 17]=  0      p[ 18]=  0      p[ 19]=  0
p[ 20]=  0      p[ 21]=  0      p[ 22]=  0      p[ 23]=  0
p[ 24]=  0      p[ 25]=  0      p[ 26]=  0      p[ 27]=  0
p[ 28]=  0      p[ 29]=  0      p[ 30]=  0      p[ 31]=  0
p[ 32]=  0      p[ 33]=  0      p[ 34]=  0      p[ 35]=  0
p[ 36]=  0      p[ 37]=  0      p[ 38]=  0      p[ 39]=  0
p[ 40]=241      p[ 41]=252      p[ 42]=  1      p[ 43]=  0
p[ 44]=  0      p[ 45]=  0      p[ 46]=  0      p[ 47]=  0
p[ 48]=  0      p[ 49]=  0      p[ 50]=  0      p[ 51]=  0
p[ 52]=  0      p[ 53]=  0      p[ 54]=  0      p[ 55]=  0
p[ 56]=  0      p[ 57]=  0      p[ 58]=  0      p[ 59]=  0
p[ 60]=  0
```

Inoltre, in ultimo, una eventuale istruzione di *free* sul puntatore "oldp", che già punta a una zona di memoria liberata, genera un errore *double free detected* - il programma in questo caso si interromperebbe qui, in quanto l'operazione sarebbe vietata.

In definitiva, la funzione *malloc* esegue le seguenti operazioni sulla memoria che tocca:
8 byte prima del primo elemento allocato, viene inserito un contatore di quanta memoria in byte è stata occupata per questo puntatore.
Alla fine dello spazio riservato, vengono inseriti 3 byte di informazione: in ordine, quanta ne rimane nel blocco corrente occupato, quanti blocchi disponibili sono ancora presenti, e un valore 1 finale di delimitazione.