F02 - Listor, dynamiskt minne

5DV149 Datastrukturer och algoritmer
Kapitel 3-4

Niclas Börlin
niclas.borlin@cs.umu.se

2020-01-23 Thu

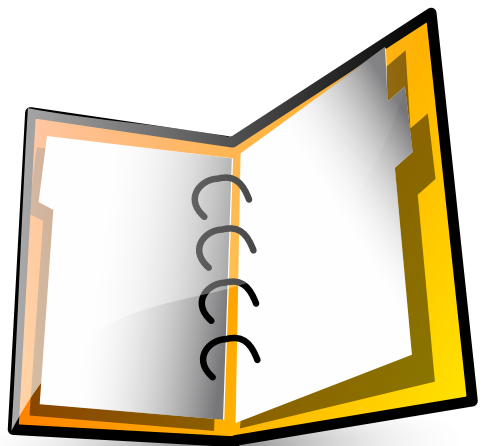
Listor (abstrakta)

Dynamiska resurser

- Skapar dataobjekt för tillfälliga behov.
- `Create` — reserverar resurser.
  - Jämför med `malloc`, `calloc` i C.
- `Kill` — återlämning av resurser.
  - Jämför med `free` i C.
- Vålnader (*dangling pointers*)
  - Minne som är avallokerat, men som vi fortfarande refererar till.
- En del språk har explicit återlämning av minne.
  - Ex. C
- Andra språk har s.k. implicit återlämning av minne.
  - Ex. Python.
  - Periodvis körs sophämtning (*garbage collection*).

Lista (1)

- Modell: Pärm.
  - Bläddra, inspektera, lägga till, ta bort.
  - Vi kan lätt ta oss till början eller slutet.
  - Vi kan röra oss framåt och bakåt.
- Konstruktion:
  - Dynamiskt med hjälp av länkade celler.
  - Statiskt med hjälp av fält/arrayer.
- Byggblock för många andra strukturer.



Lista (2)

- Sammansatt datatyp — lagrar *element*.
- Generisk datatyp (polytyp)
  - Lista av <Typ>
    - Typ kan vara av vilken typ som helst
- Homogen datatyp:
  - Alla element har samma typ.
- Elementen är *linjärt ordnade*
  - Elementen följer en före/efter-relation.
- Exempel:
  - Lista av heltal `( 3 12 -7 23 5 ).`
  - Lista av tecken `( a x p / 0 ! ).`
  - Lista av (Lista av heltal) `( (5 2 9) (44 1) (2 0 9) )`

Lista (3)

- Ändligt antal linjärt ordnade element.
  - Första / sista element.
  - Före / efter relation.
  - Alla element utom det sista har en unik efterföljare.
  - Alla element utom det första har en unik föregångare.
- Dynamisk datatyp:
  - Struktur och storlek förändras under datatypens livslängd.

Lista (4)

- Varje element har två egenskaper:
  - Ett värde.
  - En position.
- Position:
  - En plats i strukturen.
  - Viktigt: För en lista med n element, finns $n + 1$ positioner!
  - Den sista positionen i listan är **efter** det sista elementet!
- Struktur:
  - Förändras vid insättning och borttagning.
  - Oberoende av elementens **värden**.
- Exempel: Följande är två listor med 3 element:
  - `( 3 12 -7 ).`
  - `( (5 2 9) (44 1) (2 0 9) )`

Länk

- För att bygga upp listor behöver vi länkar.
  - Referens, pekare.
  - Objekt som refererar till annat objekt.
- Är vanligen pekare i språket.
- Kan också vara t.ex. index i fält.
- Billigare att kopiera länkar till objekt än objekten själva.

Gränsyta till Lista

```
abstract datatype List(val)
auxiliary pos
  Empty() → List(val)
  Isempty (l: List(val)) → Bool
  First (l: List(val)) → pos
  End (l: List(val)) → pos
  Next (p: pos, l: List(val)) → pos
  Previous (p: pos, l: List(val)) → pos
  Inspect (p: pos, l: List(val)) → val
  Insert(v: val, p: pos, l: List(val))
                                  → (List(val),pos)
  Remove(p: pos, l: List(val)) → (List(val),pos)
```

Begränsningar

```
Next (p: pos, l: List(val)) → pos
Previous (p: pos, l: List(val)) → pos
Inspect (p: pos, l: List(val)) → val
Remove(p: pos, l: List(val)) → (List(val),pos)
```

- `Next` *odefinierad* för positionen `End`
- `Previous` *odefinierad* för positionen `First`.
- `Inspect` *odefinierad* för positionen `End`.
- `Remove` *odefinierad* för positionen `End`.

Förtydliganden

```
Insert(v: val, p: pos, l: List(val))
                                  → (List(val),pos)
Remove(p: pos, l: List(val)) → (List(val),pos)
```

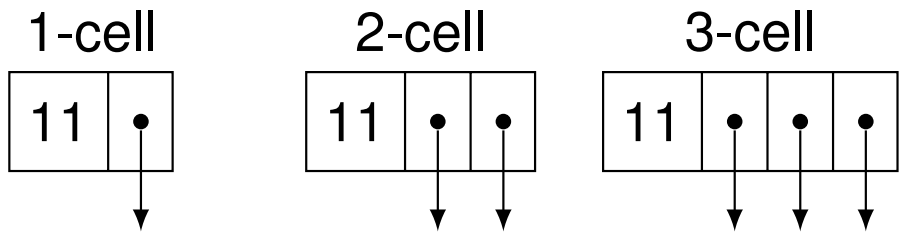
- `Insert` sätter in *v* på positionen *omedelbart före* *p* och returnerar *positionen* för det nyinsatta elementet.
- `Remove` tar bort elementet på positionen *p* och returnerar positionen *omedelbart efter* det borttagna elementet.

Positioner, varning

- En position i en lista är bara giltig så länge som listan inte modifieras!
- Till exempel:
  - `pos1 = Next( First(l), l)`
  - `(l, pos2) = Insert(24, First(l), l)`
- Efter anropet till `Insert` så är `pos1` ej garanterad att vara en korrekt position i listan.

n-länkad Cell

- En Tippel bestående av:
  - Ett värde (kan vara en länk).
  - *n* stycken länkar.



- Byggmaterial för andra datatyper.
- *n*-länkad struktur:
  - Objekt konstruerade med *n*-länkade celler:
    - Listor (*next*, *previous*), träd (*left*, *right*), . . .
- Kan gömmas inuti implementationen av Lista.

Illustration av celler

- En nil-pekare illustreras av en tom låda eller en jord-symbol.
- Röda element är osynliga för den som använder datatypen.

Gränsyta till 1Cell — Cell med en länk

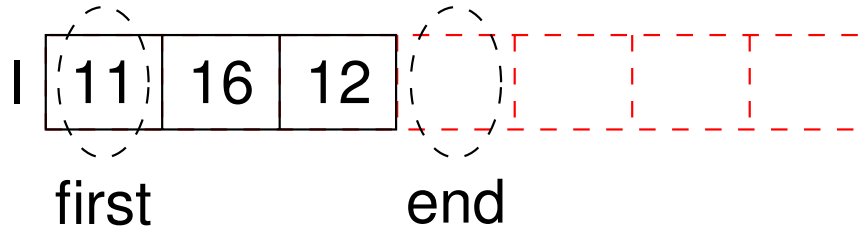
```
abstract datatype 1Cell(val)
  Create() → 1Cell(val)
  Kill(c: 1Cell(val))
  Set-value (v: val, c: 1Cell(val)) → 1Cell(val)
  Set-link (l: Link(1Cell(val)), c: 1Cell(val))
                                    → 1Cell(val)
  Inspect-value (c: 1Cell(val)) → val
  Inspect-link (c: 1Cell(val)) → Link(1Cell(val))
```

Gränsyta till 2Cell — Cell med två länkar

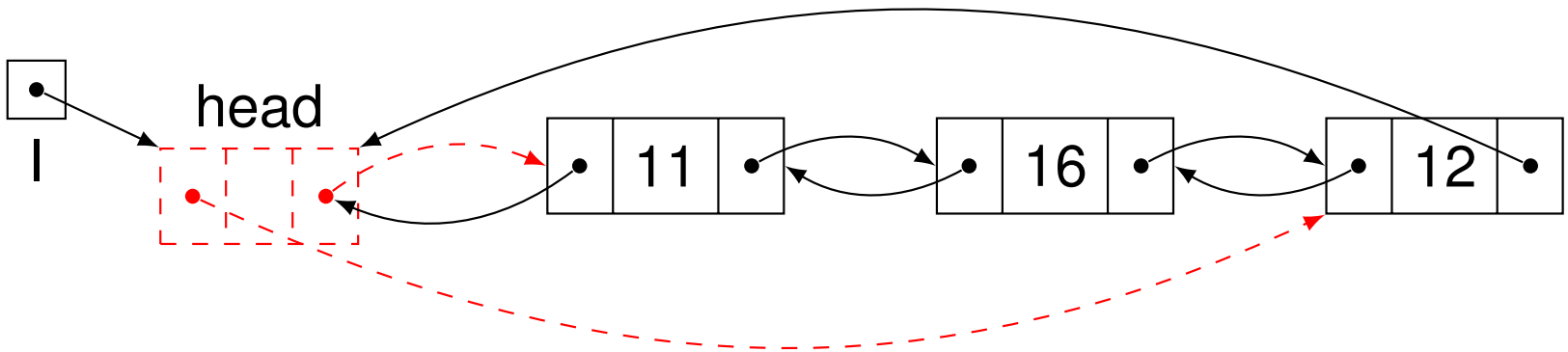
```
abstract datatype 2Cell(val)
  Create() → 2Cell(val)
  Kill(c: 2Cell(val))
  Set-value (v: val, c: 2Cell(val)) → 2Cell(val)
  Set-link1 (l: Link(2Cell(val)), c: 2Cell(val))
                                    → 2Cell(val)
  Set-link2 (l: Link(2Cell(val)), c: 2Cell(val))
                                    → 2Cell(val)
  Inspect-value (c: 2Cell(val)) → val
  Inspect-link1 (c: 2Cell(val)) → Link(2Cell(val))
  Inspect-link2 (c: 2Cell(val)) → Link(2Cell(val))
```

Sätt att konstruera lista (1)

- Lista som fält:

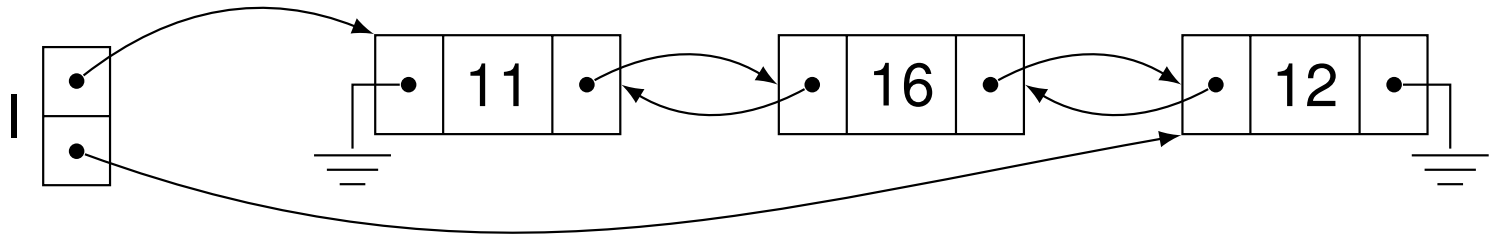


  - Positionstypen är samma som indextypen (vanligen heltal).
- Lista konstruerad med 2-Cell (dubbellänkad lista) med 2-Cell-huvud:

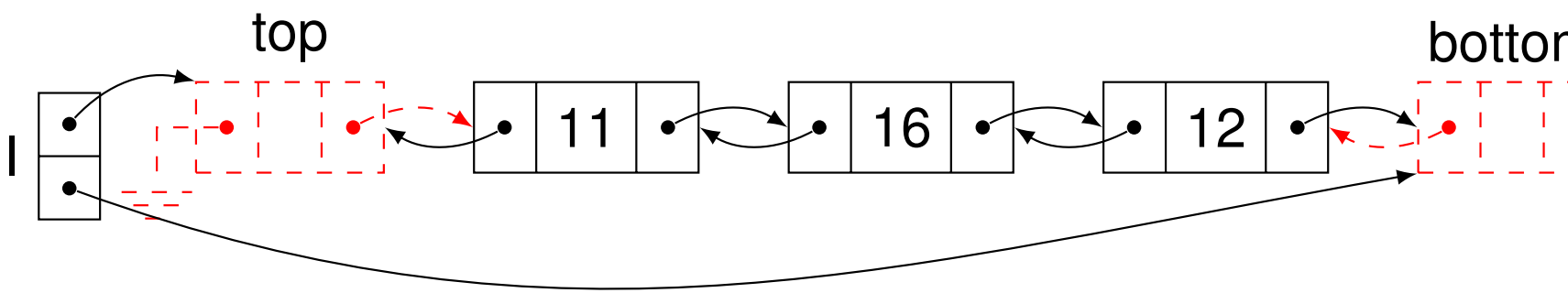


Sätt att konstruera lista (2)

- Lista konstruerad med 2-Cell och annan listhuvudstyp:



- Lista konstruerad med 2-Cell, annan listhuvudstyp och före-efter-celler:



Två exempel på `List`- och `Pos`-typer i C (1)

code/int_list_2cell.h

```
6   typedef struct two_cell {
7       struct two_cell *next;
8       struct two_cell *previous;
9       int value;
10  } two_cell;
11
12  typedef two_cell* list_position;
13
14  typedef struct {
15      two_cell *top;
16      two_cell *bottom;
17  } list;
```

Två exempel på `List`- och `Pos`-typer i C (2)

code/int_list_array.h

```
24  // List type.
25  typedef struct list list;
26
27  // List position type.
28  typedef int list_position;
```

code/int_list_array.c

```
20  /*
21   * The list is implemented as a static array.
22   */
23  struct list {
24      int last_used_pos;
25      int *values;
26  };
```

Empty, Isempty — List-2Cell

- Empty:

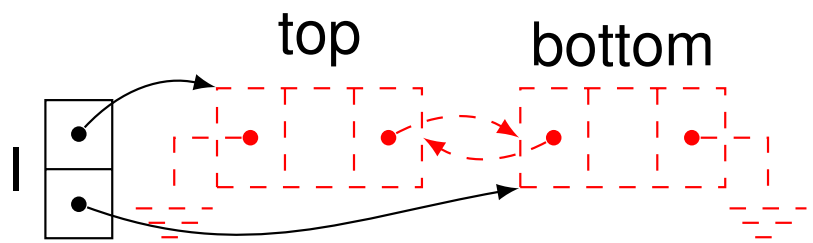
code/int_list_2cell.c

```
5  list *list_empty(void){
6      list *l=malloc(sizeof(list)) ;
7      l->top=calloc(1,sizeof(two_cell));
8      l->bottom=calloc(1,sizeof(two_cell));
9      l->top->next=l->bottom;
10     l->bottom->previous=l->top;
11     return l;
12 }
```

- Isempty:

code/int_list_2cell.c

```
14 bool list_isEmpty(const list *l){
15     return (l->top->next==l->bottom);
16 }
```



Empty, Isempty — List-Array

- Empty:

code/int_list_array.c

```
39  list *list_empty(void)
40  {
41      // Allocate memory for the list head.
42          list *l=malloc(sizeof(list));
43          // Allocate memory for the elements.
44          l->values=calloc(ARRAY_MAX_SIZE,sizeof(int));
45          // Set last used position.
46          l->last_used_pos=-1;
47          return l;
48  }
```

- Isempty:

code/int_list_array.c

```
56  bool list_is_empty(const list * l)
57  {
58      // List is empty if no elements are used.
59      return l->last_used_pos<0;
60  }
```

Navigering — List-2Cell (1)

- First:

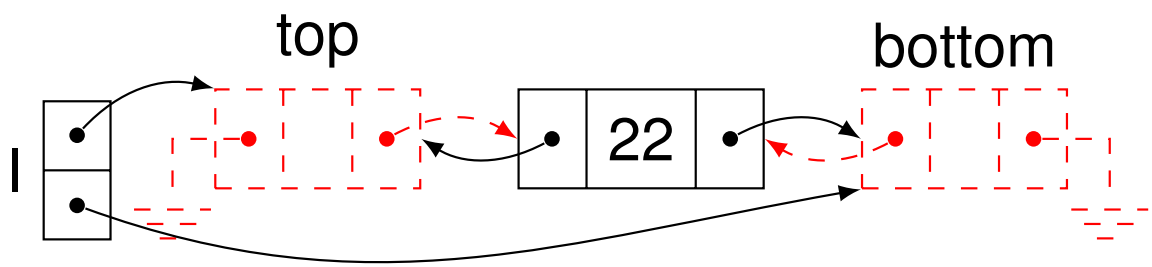
code/int_list_2cell.c

```
22  list_position list_first(const list *l){
23      return l->top->next;
24  }
```

- End:

code/int_list_2cell.c

```
26  list_position list_end(const list *l){
27      return l->bottom;
28  }
```



Navigering — List-Array (1)

- First:

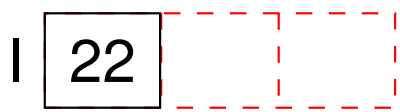
code/int_list_array.c

```
69  list_position list_first(const list * l)
70  {
71      // First position is always 0.
72      return 0;
73  }
```

- End:

code/int_list_array.c

```
82  list_position list_end(const list * l)
83  {
84      // Last position is position *after* last used element.
85      return l->last_used_pos + 1;
86  }
```



Navigering — List-2Cell (2)

- Next:

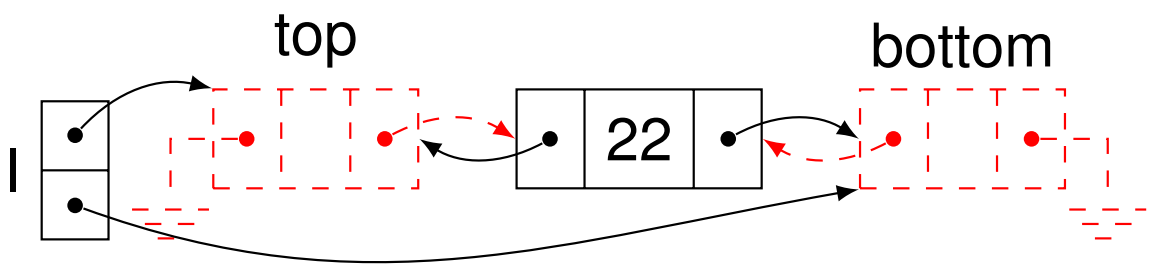
code/int_list_2cell.c

```
30 list_position list_next(const list *l,const list_position p){
31     return p->next;
32 }
```

- Previous:

code/int_list_2cell.c

```
34 list_position list_previous(const list *l,const list_position p){
35     return p->previous;
36 }
```



Navigering — List-Array (2)

- Next:

code/int_list_array.c

```
96  list_position list_next(const list * l, const list_position pos)
97  {
```

code/int_list_array.c

```
103     return pos + 1;
104 }
```

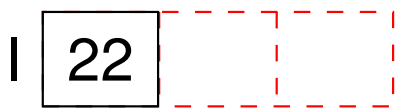
- Previous:

code/int_list_array.c

```
114 list_position list_previous(const list * l, const list_position pos)
115 {
```

code/int_list_array.c

```
121     return pos - 1;
122 }
```

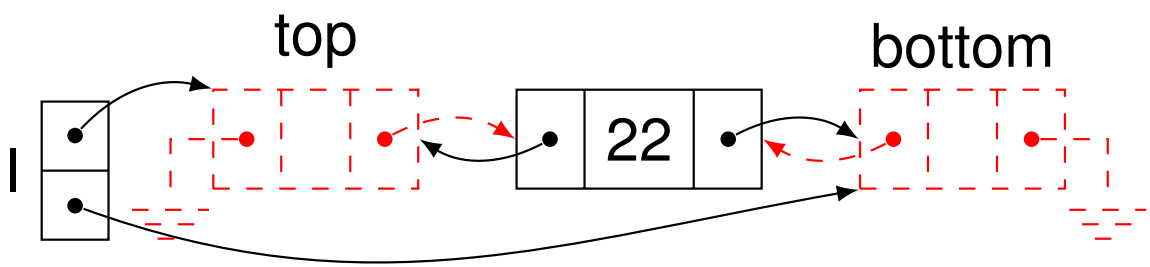


Inspect — List-2Cell

- Inspect:

code/int_list_2cell.c

```
18  int list_inspect(const list *l,const list_position pos){
19      return pos->value;
20  }
```



Inspect — List-Array

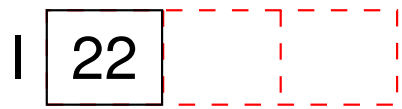
- Inspect:

code/int_list_array.c

```
133 int list_inspect(const list * l, const list_position pos)
134 {
```

code/int_list_array.c

```
140     return l->values[pos];
141 }
```

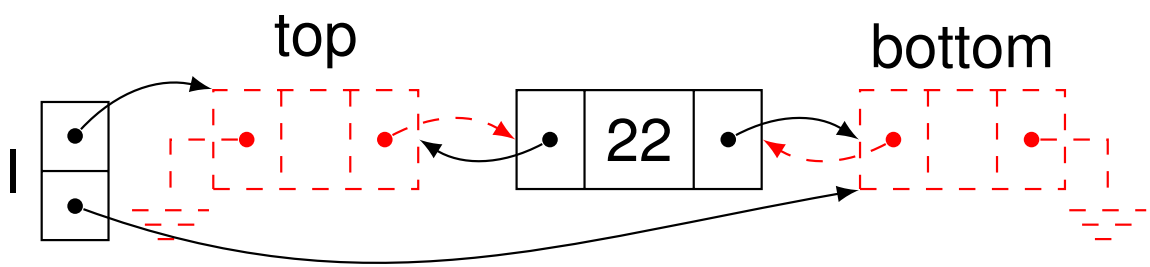


Modifikatorer — List-2Cell (1)

- Insert:

code/int_list_2cell.c

```
38 list_position list_insert(list *l,int value, const list_position p){
39     list_position newlink=malloc(sizeof(two_cell));
40     newlink->value=value;
41     newlink->next=p;
42     newlink->previous=p->previous;
43     p->previous=newlink;
44     newlink->previous->next=newlink;
45     return newlink;
46 }
```

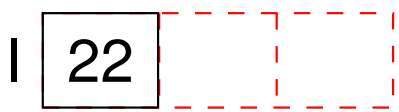


Modifikatorer — List-Array (1)

- Insert:

code/int_list_array.c

```
154 list_position list_insert(list * l, int data, const list_position pos)
155 {
156     // Move elements at position pos and later forward.
157     bcopy(l->values+pos,l->values+pos+1,
158               sizeof(int)*(l->last_used_pos-pos+1));
159
160     // Set value.
161     l->values[pos]=data;
162
163     // Increment number of used elements.
164     l->last_used_pos++;
165
166     // Return the position of the new cell.
167     return pos;
168 }
```

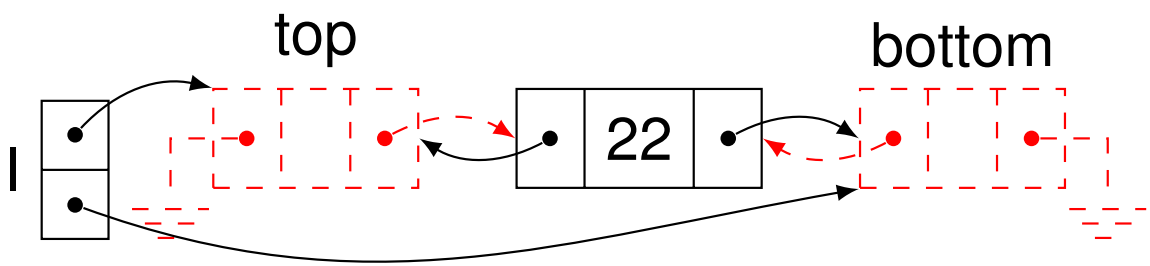


Modifikatorer — List-2Cell (2)

- Remove:

code/int_list_2cell.c

```
48  list_position list_remove(list *l,const list_position p){
49      list_position retur=p->next;
50      p->previous->next=p->next;
51      p->next->previous=p->previous;
52      free(p);
53      return retur;
54  }
```

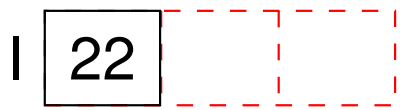


Modifikatorer — List-Array (2)

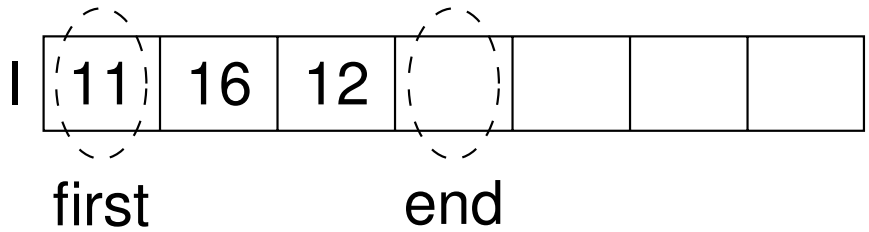
- Remove:

code/int_list_array.c

```
181 list_position list_remove(list * l, const list_position pos)
182 {
183     // Move elements at position pos and later forward.
184     bcopy(l->values+pos+1,l->values+pos,sizeof(int)*(l->last_used_pos-pos));
185
186     // Decrement number of used elements.
187     l->last_used_pos--;
188
189     // Return the position of the next element.
190     return pos+1;
191 }
```

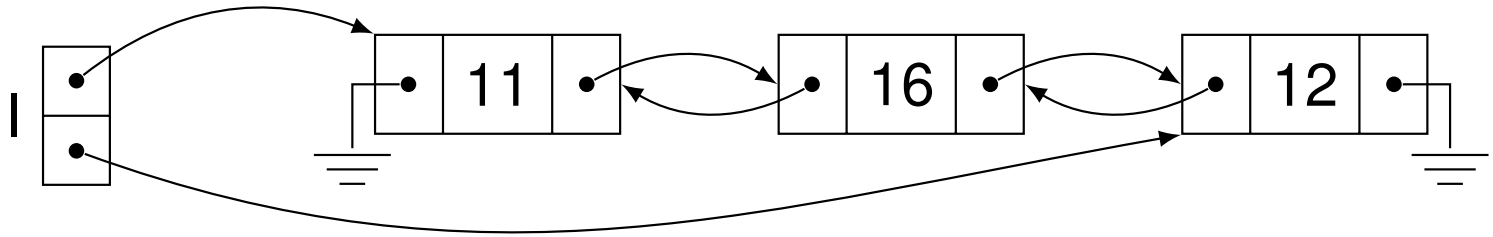


Jämförelse — Lista som Fält



- Fördelar:
  - Snabb inspektion av alla element.
- Nackdelar:
  - Fast reserverat utrymme.
  - Kostsamt sätta in/ta bort element om element måste flyttas.

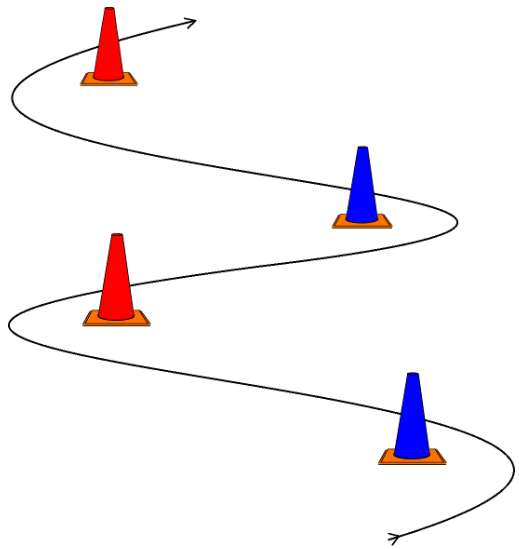
Jämförelse — Lista som Länkad struktur



- Fördelar:
  - Insättning/borttagning går snabbt.
  - Minnesutrymmet är proportionellt mot storleken på listan.
  - Allokerar minne när det behövs.
- Nackdelar:
  - Länkarna behöver också minnesutrymme.
  - Kommer bara åt listelement genom att traversera från listans början eller slut.

Riktad lista

- Modell: Slalombana, skattjakt.
- Vi kan ta oss till starten.
- Inom listan kan vi bara röra oss framåt.
- Specialisering av Lista.



Gränsyta till Riktad lista (*Directed List)*

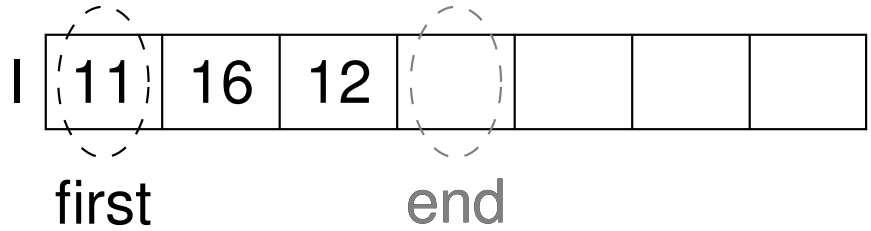
```
abstract datatype DList(val)
auxiliary pos
  Empty() → DList(val)
  Isempty (l: DList(val)) → Bool
  First (l: DList(val)) → pos
  Next (p: pos, l: DList(val)) → pos
  Isend (p: pos, l: DList(val)) → Bool
  Inspect (p: pos, l: DList(val)) → val
  Insert(v: val, p: pos, l: DList(val))
                                → (DList(val),pos)
  Remove(p: pos, l: DList(val)) → (DList(val),pos)
```

Skillnader mellan Lista och Riktad lista

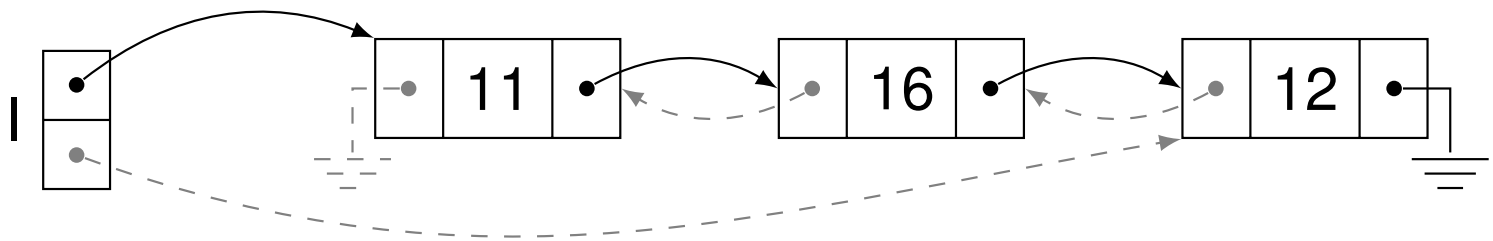
Lista	Riktad lista
`First`	`First`
`End`	
	`Isend`
`Next`	`Next`
`Previous`	
Vi kan navigera åt båda hållen	Vi kan bara navigera åt ett håll

Riktad Lista, konstruktionsalternativ (1)

- Fält:

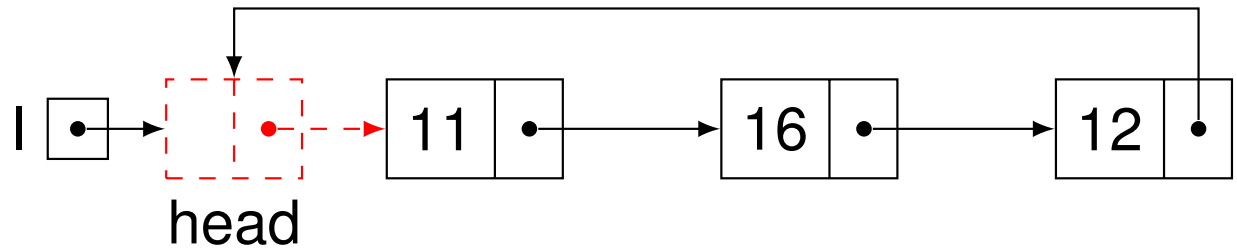


- Dubbellänkad Lista (ignorera bakåtlänkarna):

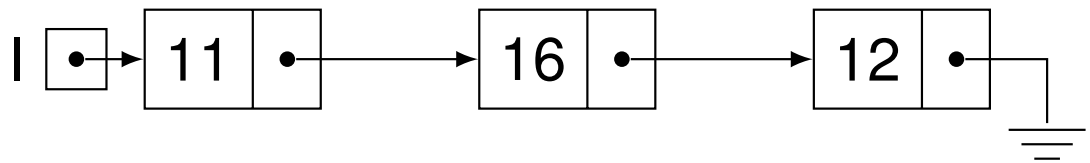


Riktad Lista, konstruktionsalternativ (2)

- Enkellänkad Lista av 1-Cell med 1-Cell-huvud:



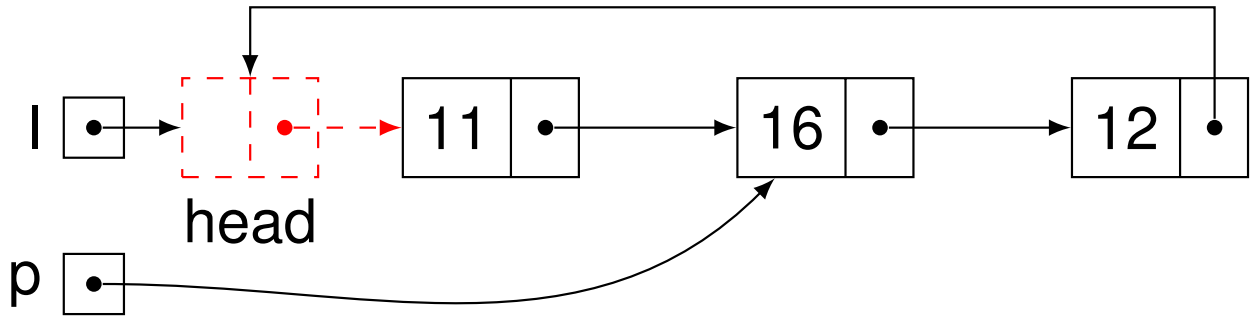
- Enkellänkad Lista av 1-Cell utan huvud:



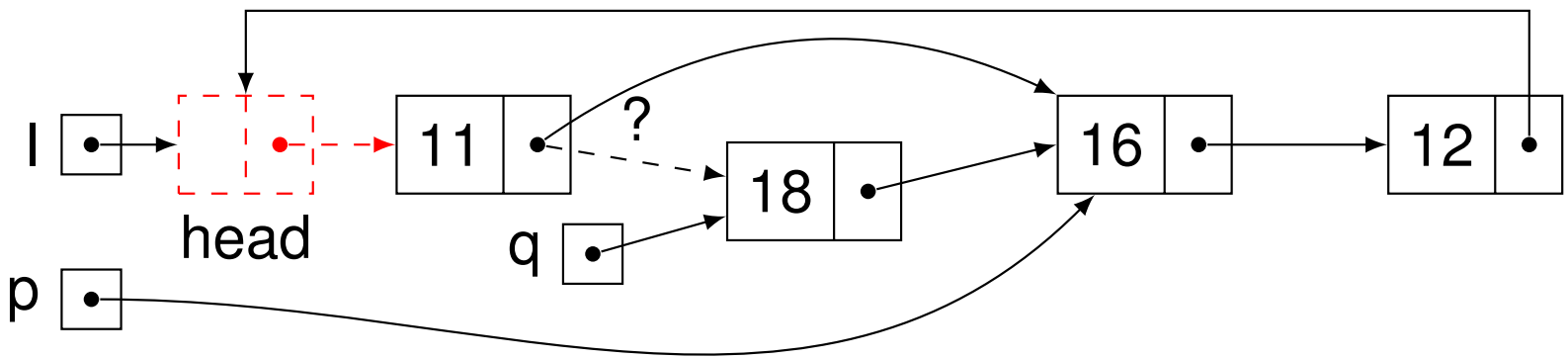
- Enkellänkad mer ekonomisk än dubbellänkad.

Enkellänkad Lista — problem vid insättning, borttagning

- Före:



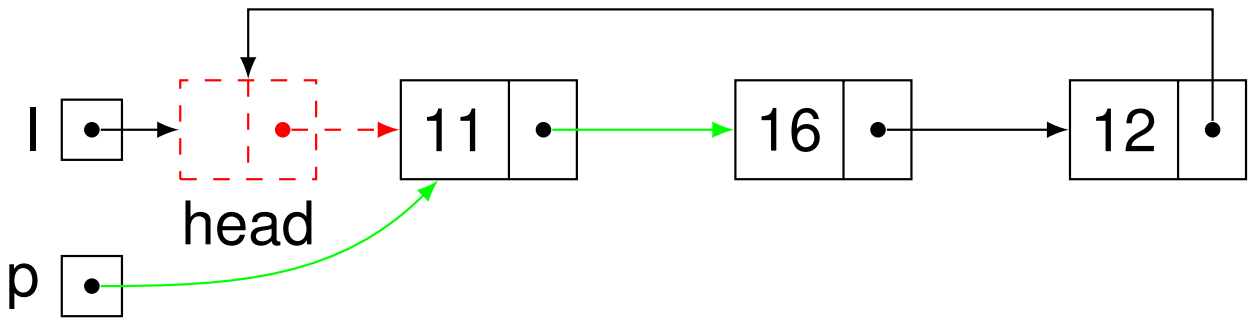
- Efter allokering av nytt element, före inlänkning.



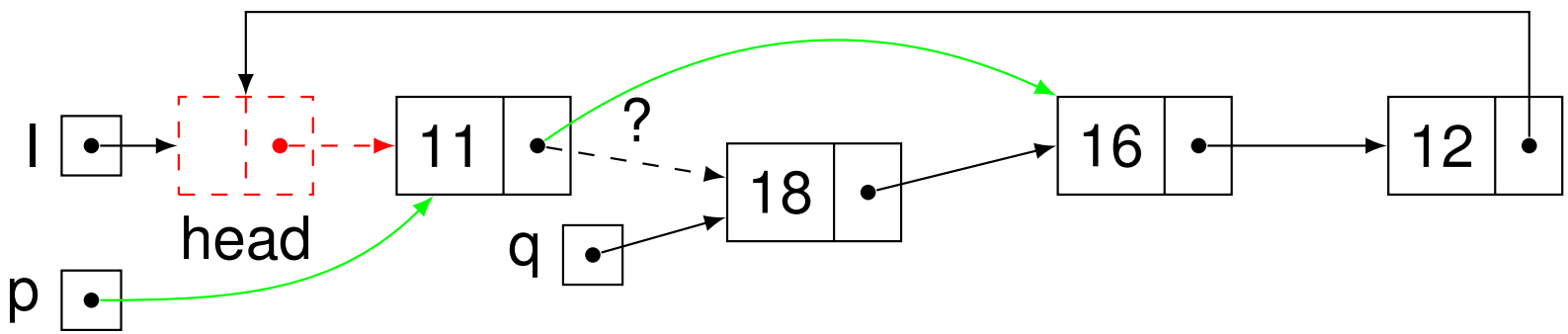
- Behöver komma åt framåt-länken i föregående cell!

Enkellänkad Lista — lösning insättning, alt. 1

- Använd list-konstruktion med huvud av samma typ som cellerna.
- Representera positionen med pekare till *föregående* element.
- Före:



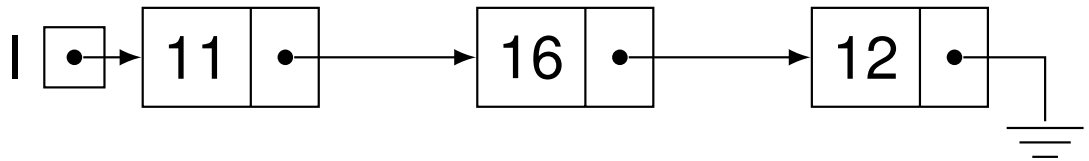
- Efter allokering av nytt element, före inlänkning.



- Nackdel: (litet) minnesslöseri.

Enkellänkad Lista — lösning insättning, alt. 2, 3

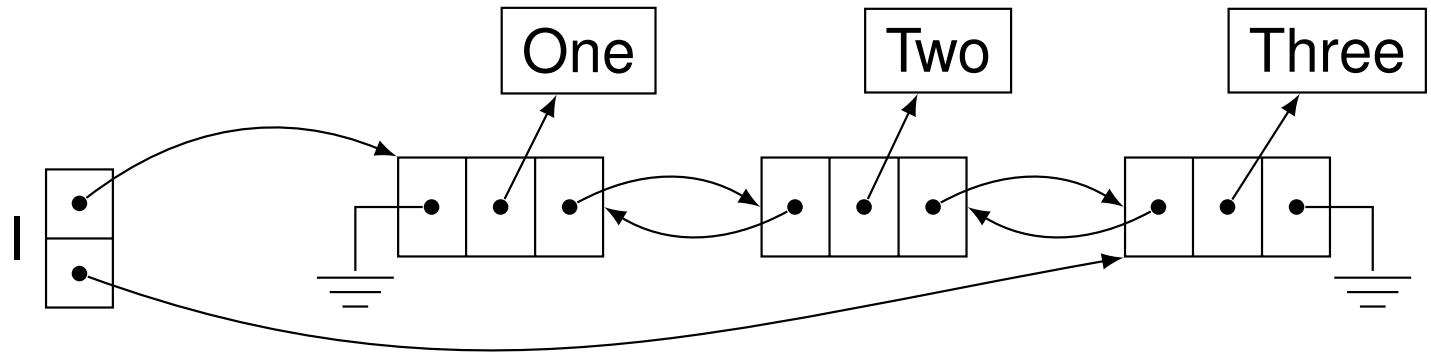
- Enkellänkad Lista av 1-Cell utan huvud:



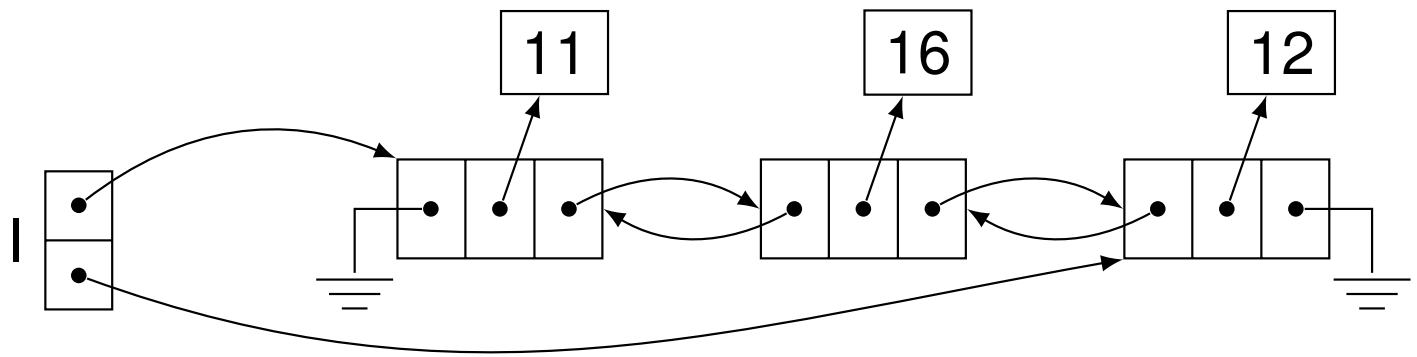
- Insättning före elementet X:
  - Skapa en ny cell.
  - Sätt in den efter X.
  - Kopiera X:s värde till den nya cellen.
  - Sätt X:s värde till v.
  - Nackdel: Värdet måste kopieras.
- Alt. traversera från huvudet till elementet före.
  - Nackdel: Insättning blir $O(n)$.

Lista av pekare

- I stället för att lagra värden i cellerna kan vi lagra Länkar till värdena.
- Lista av strängar:



- Lista av heltal:



Algoritmmönster för lista

- Traversering:
  - Besök systematiskt alla element.
- Sökning:
  - Sök efter det första elementet som uppfyller ett bestämt villkor.
- Filtrering:
  - Filtrera ut alla element som uppfyller ett bestämt villkor.
- Reduktion:
  - Beräknar en funktion av objektets elementvärden.
    - Ex. Summera alla tal i en lista.
- Avbildning (*mapping*)
  - Transformera varje elementvärde i en datastruktur:
    - Ex. multiplicera alla talen med 2.

Dynamiskt minne i C

Histogram

- Antag vi vill beräkna och skriva ut ett histogram för en vektor av värden.
- Ett histogram räknar förekomsten av varje värde.
- Exempelvis så innehåller strängen "090-786 68 32", två st nollor, ingen etta, en tvåa, osv.

Histogram med statisk minnesallokering (1)

- Om vi vet hur mycket många och stora variabler vi behöver kan vi deklarera dem **statiskt**:

code/dcount_indexed.c

```
23  int main(int argc, const char *argv[])
24  {
25      const char *msg = "090-786 68 32";
26      int n = 10;
27      int hist[n]; /* Här reserveras minnet. */
28
29      zero_histogram(hist);
30      make_histogram(msg, hist);
31      print_histogram(hist);
32
33      return 0;     /* Här återlämnas minnet. */
34  }
```

Histogram med statisk minnesallokering (2)

code/dcount_indexed.c

```c
#include <stdio.h>

void zero_histogram(int *hist)
{
    for (int i = 0; i < 10; i++)
        hist[i] = 0;
}

void make_histogram(const char *msg, int *hist)
{
    for (int i = 0; msg[i] != '\0'; i++)
        if (msg[i] >= '0' && msg[i] <= '9')
            hist[msg[i] - '0']++;
}

void print_histogram(const int *hist)
{
    for (int i = 0; i < 10; i++)
        if (hist[i] > 0)
            printf("%d: %d\n", i, hist[i]);
}
```

Begränsningar med statisk minnesallokering

- Vi måste veta storleken på våra fält (*arrayer*) när vi skriver programmet (*compile time*).
  - Svårt att skriva en funktion som skapar ett histogram av variabel storlek.
- Det statiska minnet är bara reserverat tills funktionen returnerar.

Dynamisk minnesallokering

- En annan lösning är att använda **dynamisk** minnesallokering:
  - Allokera (reservera) minne när det behövs.
  - Frigör (lämna tillbaka) minne när man är klar.
- Fördelar: Ökad makt
  - Behöver inte veta storleken i förväg.
  - Vi kan använda minnet så länge vi behöver det.
- Nackdelar: Ökat ansvar
  - Vi måste komma ihåg att återlämna minnet när vi är klara.
  - Vi får inte använda minnet när vi lämnat tillbaka det.

Histogram med dynamisk minnesallokering

- Så här ser motsvarande huvudprogram ut med dynamisk minnesallokering:

code/dcount_dynamic.c

```
24  int main(int argc, const char *argv[])
25  {
26      const char *msg = "090-786 68 32";
27      int n = 10;
28      int *hist = malloc(n * sizeof(int)); /* reserveration */
29
30      zero_histogram(hist);
31      make_histogram(msg, hist);
32      print_histogram(hist);
33
34      free(hist);                          /* återlämning */
35      return 0;
36  }
```

- Övrig kod är identisk.

Dynamisk minneshantering i C (1)

- För att allokera minne i C används de inbyggda funktionerna `malloc` och `calloc`:

stdlib.h

```
void *malloc(size_t size);
void *calloc(size_t nelem, size_t size);
```

- Bägge returnerar en pekare till allokerat minne om OK, annars `NULL`.
- Typen **`size_t`** är definierad till **`unsigned int`** i ANSI C.
- `malloc` allokerar `size` *bytes* med minne.
- `calloc` allokerar `nelem` element av storleken `size` bytes.
- `calloc` initierar dessutom det allokerade minnets alla bitar till 0.

Dynamisk minneshantering i C (2)

- För att allokera minne till en heltalsvektor med 10 element, använd någon av

```
int *v1 = malloc(10*sizeof(int));
int *v2 = calloc(10, sizeof(int));
int *v3 = malloc(10*sizeof(*v3));
int *v4 = calloc(10, sizeof(*v4));
```

- Operatorn **`sizeof`** returnerar storleken på argumentet i bytes.

Dynamisk minneshantering i C (3)

- Funktionen `free` används för att explicit lämna tillbaka minne.

stdlib.h

```
void free(void* ptr);
```

- Not: `free` behöver bara pekaren, inte storleken.

code/dcount_dynamic.c

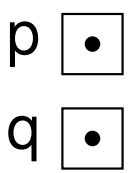
```
24  int main(int argc, const char *argv[])
25  {
26      const char *msg = "090-786 68 32";
27      int n = 10;
28      int *hist = malloc(n * sizeof(int)); /* reserveration */
29
30      zero_histogram(hist);
31      make_histogram(msg, hist);
32      print_histogram(hist);
33
34      free(hist);                          /* återlämning */
35      return 0;
36  }
```

Aliasing

- Allokerat minne är inte knutet till en specifik pekarvariabel.
- Två variabler kan referera till samma minne, s.k. *aliasing*.

code/multiptr.c

```
1  #include <stdio.h>
2  #include <stdlib.h>
3
4  int main(void)
5  {
6      char *p, *q;
7      p = malloc(5);
8      /* do stuff */
9      q = p;
10     free(q);     /* this is ok */
11     return 0;
12 }
```

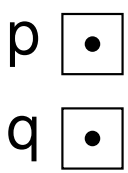


Aliasing

- Allokerat minne är inte knutet till en specifik pekarvariabel.
- Två variabler kan referera till samma minne, s.k. *aliasing*.

code/multiptr.c

```
1  #include <stdio.h>
2  #include <stdlib.h>
3
4  int main(void)
5  {
6      char *p, *q;
7      p = malloc(5);
8      /* do stuff */
9      q = p;
10     free(q);     /* this is ok */
11     return 0;
12 }
```

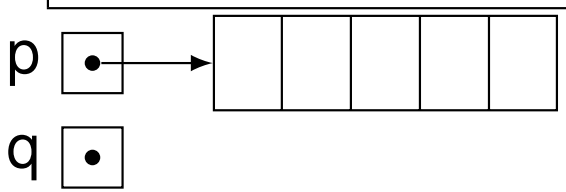


Aliasing

- Allokerat minne är inte knutet till en specifik pekarvariabel.
- Två variabler kan referera till samma minne, s.k. *aliasing*.

code/multiptr.c

```
1  #include <stdio.h>
2  #include <stdlib.h>
3
4  int main(void)
5  {
6      char *p, *q;
7      p = malloc(5);
8      /* do stuff */
9      q = p;
10     free(q);     /* this is ok */
11     return 0;
12 }
```

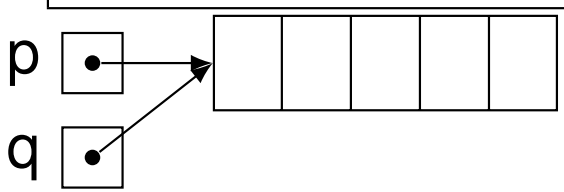


Aliasing

- Allokerat minne är inte knutet till en specifik pekarvariabel.
- Två variabler kan referera till samma minne, s.k. *aliasing*.

code/multiptr.c

```
1   #include <stdio.h>
2   #include <stdlib.h>
3
4   int main(void)
5   {
6       char *p, *q;
7       p = malloc(5);
8       /* do stuff */
9       q = p;
10      free(q);     /* this is ok */
11      return 0;
12  }
```

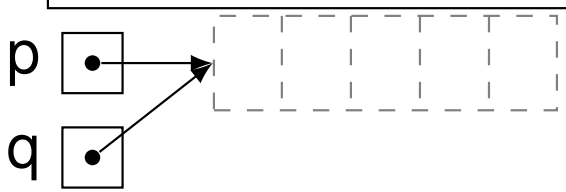


Aliasing

- Allokerat minne är inte knutet till en specifik pekarvariabel.
- Två variabler kan referera till samma minne, s.k. *aliasing*.

code/multiptr.c

```
1  #include <stdio.h>
2  #include <stdlib.h>
3
4  int main(void)
5  {
6      char *p, *q;
7      p = malloc(5);
8      /* do stuff */
9      q = p;
10     free(q);     /* this is ok */
11     return 0;
12 }
```

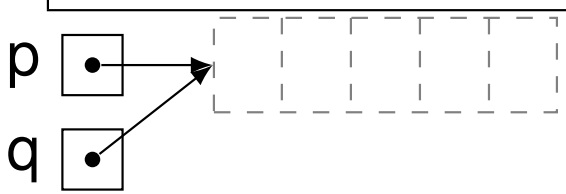


Aliasing

- Allokerat minne är inte knutet till en specifik pekarvariabel.
- Två variabler kan referera till samma minne, s.k. *aliasing*.

code/multiptr.c

```
1  #include <stdio.h>
2  #include <stdlib.h>
3
4  int main(void)
5  {
6      char *p, *q;
7      p = malloc(5);
8      /* do stuff */
9      q = p;
10     free(q);     /* this is ok */
11     return 0;
12 }
```

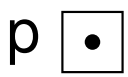


Minnesläckor

- **Om man förlorar referensen till allokerat minne kan man inte sedan referera det eller frigöra det.**

code/lostref.c

```
1  #include <stdio.h>
2  #include <stdlib.h>
3
4  int main(void)
5  {
6      char *p;
7      p = malloc(5);
8      /* do stuff, forget to call free */
9      p = malloc(7);
10     free(p);
11     return 0;
12 }
```

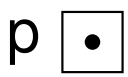


Minnesläckor

- **Om man förlorar referensen till allokerat minne kan man inte sedan referera det eller frigöra det.**

code/lostref.c

```
1  #include <stdio.h>
2  #include <stdlib.h>
3
4  int main(void)
5  {
6      char *p;
7      p = malloc(5);
8      /* do stuff, forget to call free */
9      p = malloc(7);
10     free(p);
11     return 0;
12 }
```

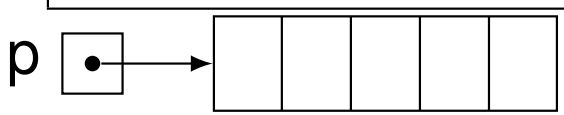


Minnesläckor

- **Om man förlorar referensen till allokerat minne kan man inte sedan referera det eller frigöra det.**

code/lostref.c

```
1  #include <stdio.h>
2  #include <stdlib.h>
3
4  int main(void)
5  {
6      char *p;
7      p = malloc(5);
8      /* do stuff, forget to call free */
9      p = malloc(7);
10     free(p);
11     return 0;
12 }
```

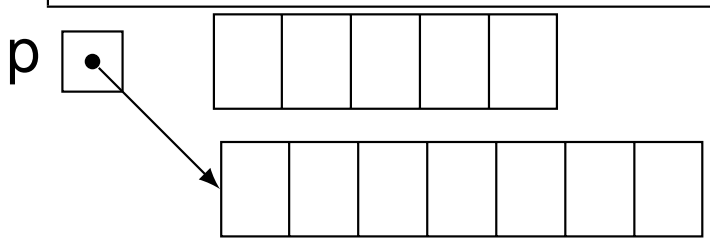


Minnesläckor

- **Om man förlorar referensen till allokerat minne kan man inte sedan referera det eller frigöra det.**

code/lostref.c

```
1  #include <stdio.h>
2  #include <stdlib.h>
3
4  int main(void)
5  {
6      char *p;
7      p = malloc(5);
8      /* do stuff, forget to call free */
9      p = malloc(7);
10     free(p);
11     return 0;
12 }
```

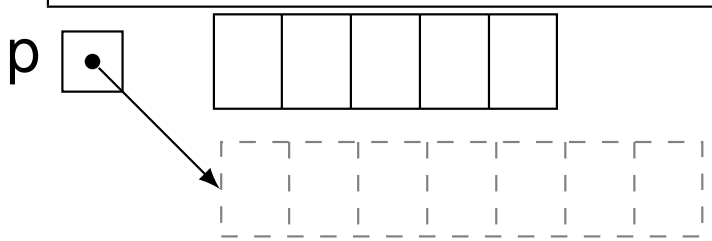


Minnesläckor

- **Om man förlorar referensen till allokerat minne kan man inte sedan referera det eller frigöra det.**

code/lostref.c

```
1  #include <stdio.h>
2  #include <stdlib.h>
3
4  int main(void)
5  {
6      char *p;
7      p = malloc(5);
8      /* do stuff, forget to call free */
9      p = malloc(7);
10     free(p);
11     return 0;
12 }
```

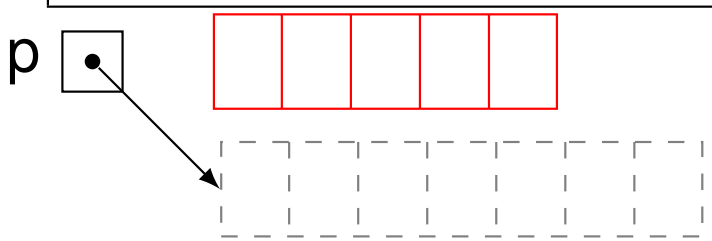


Minnesläckor

- **Om man förlorar referensen till allokerat minne kan man inte sedan referera det eller frigöra det.**

code/lostref.c

```
1  #include <stdio.h>
2  #include <stdlib.h>
3
4  int main(void)
5  {
6      char *p;
7      p = malloc(5);
8      /* do stuff, forget to call free */
9      p = malloc(7);
10     free(p);
11     return 0;
12 }
```



Ansvar för återlämning

- Extra viktigt vara **tydlig i dokumentationen** av funktioner som allokerar minne!
- Vem har **ansvar** för att avallokera?

code/lostmem.c

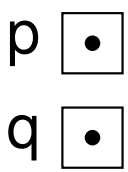
```
1  #include <stdio.h>
2  #include <stdlib.h>
3
4  int *new_int_array(int n)
5  {
6      int *v;
7      v = malloc(n * sizeof(int));
8      return v;
9  }
10
11 int main(void)
12 {
13     int *p, *q;
14     p = new_int_array(10);
15     /* Do stuff */
16     q = new_int_array(10);
17     /* Do more stuff */
18     /* No deallocation! */
19     return 0;
20 }
```

Vålnader (*dangling pointers*)

- **Referera inte till pekare efter `free`!**

code/ghostptr.c

```
4   int main(void)
5   {
6       char *p, *q;
7       p = malloc(5);
8       /* do stuff */
9       free(p);
10      if (*p == 0) {      /* Unsafe, but might not crash... */
11          return -1;
12      }
13      q = malloc(7);
14      if (*p == 0) {      /* This is really not what you want... */
15          return -1;
16      }
17      free(q);
18      return 0;
19  }
```

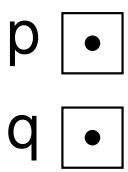


Vålnader (*dangling pointers*)

- **Referera inte till pekare efter `free`!**

code/ghostptr.c

```
4   int main(void)
5   {
6       char *p, *q;
7       p = malloc(5);
8       /* do stuff */
9       free(p);
10      if (*p == 0) {     /* Unsafe, but might not crash... */
11          return -1;
12      }
13      q = malloc(7);
14      if (*p == 0) {     /* This is really not what you want... */
15          return -1;
16      }
17      free(q);
18      return 0;
19  }
```

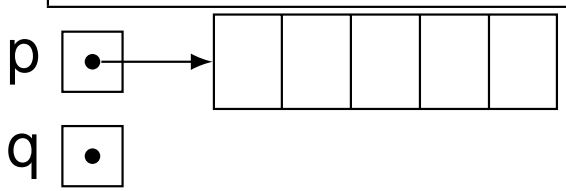


Vålnader (*dangling pointers*)

- **Referera inte till pekare efter `free`!**

code/ghostptr.c

```
4   int main(void)
5   {
6       char *p, *q;
7       p = malloc(5);
8       /* do stuff */
9       free(p);
10      if (*p == 0) {      /* Unsafe, but might not crash... */
11          return -1;
12      }
13      q = malloc(7);
14      if (*p == 0) {      /* This is really not what you want... */
15          return -1;
16      }
17      free(q);
18      return 0;
19  }
```

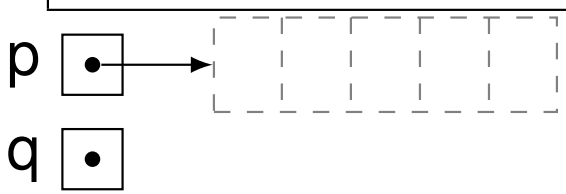


Vålnader (*dangling pointers*)

- **Referera inte till pekare efter `free`!**

code/ghostptr.c

```
4   int main(void)
5   {
6       char *p, *q;
7       p = malloc(5);
8       /* do stuff */
9       free(p);
10      if (*p == 0) {      /* Unsafe, but might not crash... */
11          return -1;
12      }
13      q = malloc(7);
14      if (*p == 0) {      /* This is really not what you want... */
15          return -1;
16      }
17      free(q);
18      return 0;
19  }
```

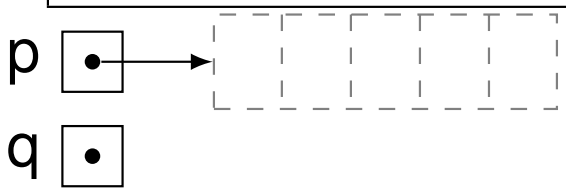


Vålnader (*dangling pointers*)

- **Referera inte till pekare efter `free`!**

code/ghostptr.c

```
4   int main(void)
5   {
6       char *p, *q;
7       p = malloc(5);
8       /* do stuff */
9       free(p);
10      if (*p == 0) {     /* Unsafe, but might not crash... */
11          return -1;
12      }
13      q = malloc(7);
14      if (*p == 0) {     /* This is really not what you want... */
15          return -1;
16      }
17      free(q);
18      return 0;
19  }
```

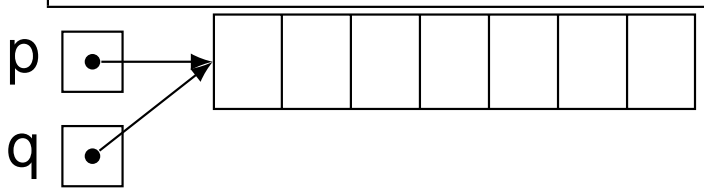


Vålnader (*dangling pointers*)

- **Referera inte till pekare efter `free`!**

code/ghostptr.c

```
4   int main(void)
5   {
6       char *p, *q;
7       p = malloc(5);
8       /* do stuff */
9       free(p);
10      if (*p == 0) {     /* Unsafe, but might not crash... */
11          return -1;
12      }
13      q = malloc(7);
14      if (*p == 0) {     /* This is really not what you want... */
15          return -1;
16      }
17      free(q);
18      return 0;
19  }
```

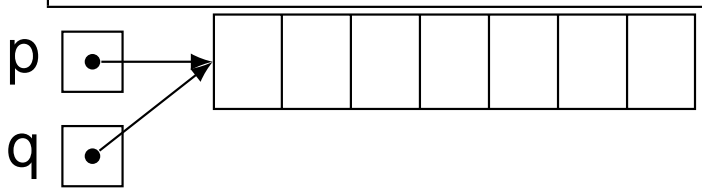


Vålnader (*dangling pointers*)

- **Referera inte till pekare efter `free`!**

code/ghostptr.c

```
4   int main(void)
5   {
6       char *p, *q;
7       p = malloc(5);
8       /* do stuff */
9       free(p);
10      if (*p == 0) {      /* Unsafe, but might not crash... */
11          return -1;
12      }
13      q = malloc(7);
14      if (*p == 0) {      /* This is really not what you want... */
15          return -1;
16      }
17      free(q);
18      return 0;
19  }
```

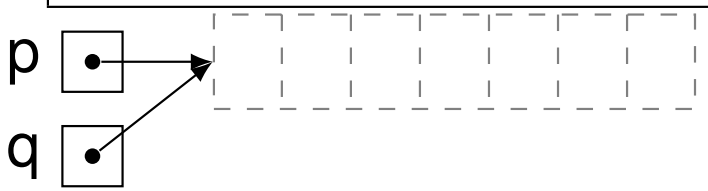


Vålnader (*dangling pointers*)

- **Referera inte till pekare efter `free`!**

code/ghostptr.c

```
4   int main(void)
5   {
6       char *p, *q;
7       p = malloc(5);
8       /* do stuff */
9       free(p);
10      if (*p == 0) {     /* Unsafe, but might not crash... */
11          return -1;
12      }
13      q = malloc(7);
14      if (*p == 0) {     /* This is really not what you want... */
15          return -1;
16      }
17      free(q);
18      return 0;
19  }
```

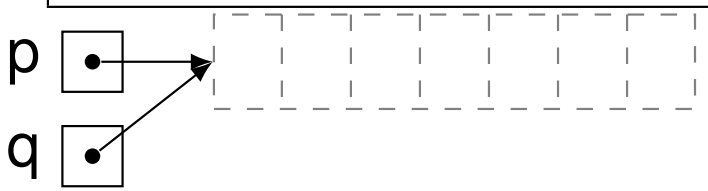


Vålnader (*dangling pointers*)

- **Referera inte till pekare efter `free`!**

code/ghostptr.c

```
4   int main(void)
5   {
6       char *p, *q;
7       p = malloc(5);
8       /* do stuff */
9       free(p);
10      if (*p == 0) {     /* Unsafe, but might not crash... */
11          return -1;
12      }
13      q = malloc(7);
14      if (*p == 0) {     /* This is really not what you want... */
15          return -1;
16      }
17      free(q);
18      return 0;
19  }
```



Dynamisk allokering av `struct`

- Kombinationen dynamiskt minne, pekare och `struct` med pekar-fält är vanlig.

code/allocstruct.c

```
5  struct Line {
6          char *string;           /* Pointer to the actual string */
7          struct Line *next;      /* Pointer to next line */
8  };
```

- Vid konstruktion så måste minne allokeras, dels till `struct`en, dels till datat.

```
10 int main(void)
11 {
12         const char *s = "Hello";
13         struct Line *p, *q;
14         p = malloc(sizeof(*p));
15         p->string = strdup(s);  /* strdup makes a dynamic copy of s */
16         p->next = NULL;
```

- Avallokering sker i omvänd ordning. Viktigt att inte tappa bort någon pekare.

```
17         free(p->string);
18         q = p->next;
19         free(p);
20         p = q;
21         return 0;
22 }
```

Dynamisk allokering av `struct`

- Kombinationen dynamiskt minne, pekare och `struct` med pekar-fält är vanlig.

code/allocstruct.c

```
5  struct Line {
6          char *string;           /* Pointer to the actual string */
7          struct Line *next;      /* Pointer to next line */
8  };
```

- Vid konstruktion så måste minne allokeras, dels till `struct`en, dels till datat.

```
10 int main(void)
11 {
12         const char *s = "Hello";
13         struct Line *p, *q;
14         p = malloc(sizeof(*p));
15         p->string = strdup(s);  /* strdup makes a dynamic copy of s */
16         p->next = NULL;
```

- Avallokering sker i omvänd ordning. Viktigt att inte tappa bort någon pekare.

```
17         free(p->string);
18         q = p->next;
19         free(p);
20         p = q;
21         return 0;
22 }
```

Dynamisk allokering av `struct`

- Kombinationen dynamiskt minne, pekare och `struct` med pekar-fält är vanlig.

code/allocstruct.c

```
5  struct Line {
6          char *string;           /* Pointer to the actual string */
7          struct Line *next;      /* Pointer to next line */
8  };
```

- Vid konstruktion så måste minne allokeras, dels till `struct`en, dels till datat.

```
10 int main(void)
11 {
12         const char *s = "Hello";
13         struct Line *p, *q;
14         p = malloc(sizeof(*p));
15         p->string = strdup(s);  /* strdup makes a dynamic copy of s */
16         p->next = NULL;
```

- Avallokering sker i omvänd ordning. Viktigt att inte tappa bort någon pekare.

```
17         free(p->string);
18         q = p->next;
19         free(p);
20         p = q;
21         return 0;
22 }
```

Dynamisk allokering av `struct`

- Kombinationen dynamiskt minne, pekare och `struct` med pekar-fält är vanlig.

code/allocstruct.c

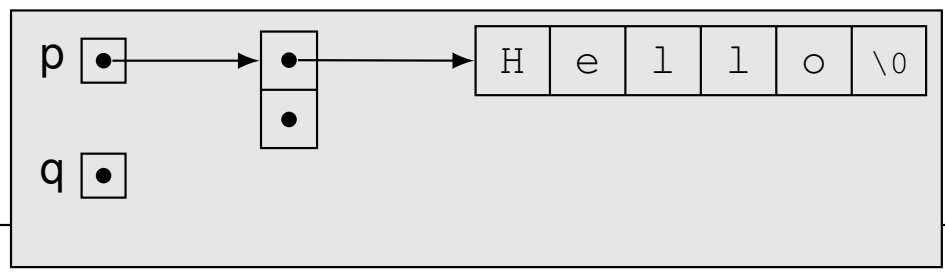
```
5  struct Line {
6          char *string;           /* Pointer to the actual string */
7          struct Line *next;      /* Pointer to next line */
8  };
```

- Vid konstruktion så måste minne allokeras, dels till `struct`en, dels till datat.

```
10 int main(void)
11 {
12         const char *s = "Hello";
13         struct Line *p, *q;
14         p = malloc(sizeof(*p));
15         p->string = strdup(s);  /* strdup makes a dynamic copy of s */
16         p->next = NULL;
```

- Avallokering sker i omvänd ordning. Viktigt att inte tappa bort någon pekare.

```
17         free(p->string);
18         q = p->next;
19         free(p);
20         p = q;
21         return 0;
22 }
```



Dynamisk allokering av `struct`

- Kombinationen dynamiskt minne, pekare och `struct` med pekar-fält är vanlig.

code/allocstruct.c

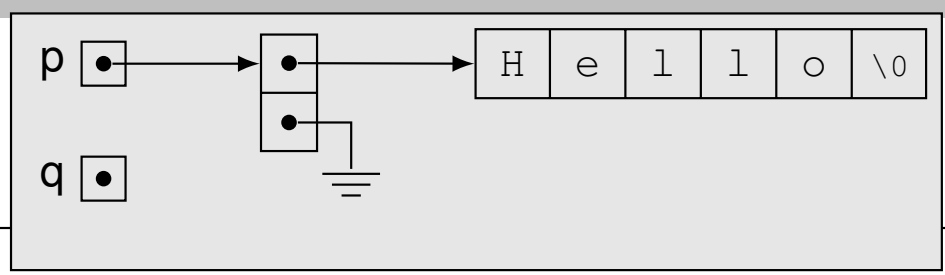
```
5  struct Line {
6          char *string;           /* Pointer to the actual string */
7          struct Line *next;      /* Pointer to next line */
8  };
```

- Vid konstruktion så måste minne allokeras, dels till `struct`en, dels till datat.

```
10 int main(void)
11 {
12         const char *s = "Hello";
13         struct Line *p, *q;
14         p = malloc(sizeof(*p));
15         p->string = strdup(s);  /* strdup makes a dynamic copy of s */
16         p->next = NULL;
```

- Avallokering sker i omvänd ordning. Viktigt att inte tappa bort någon pekare.

```
17         free(p->string);
18         q = p->next;
19         free(p);
20         p = q;
21         return 0;
22 }
```



Dynamisk allokering av `struct`

- Kombinationen dynamiskt minne, pekare och `struct` med pekar-fält är vanlig.

code/allocstruct.c

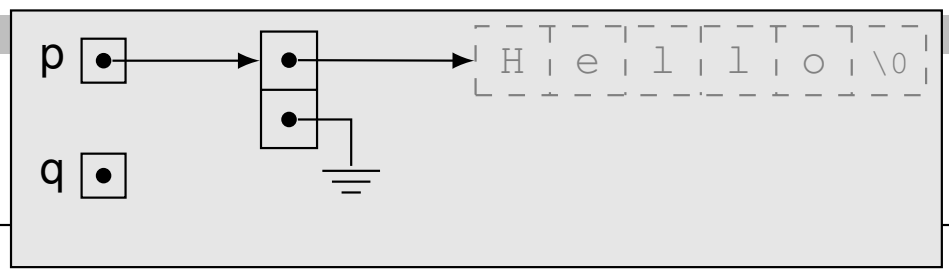
```
5  struct Line {
6          char *string;           /* Pointer to the actual string */
7          struct Line *next;      /* Pointer to next line */
8  };
```

- Vid konstruktion så måste minne allokeras, dels till `struct`en, dels till datat.

```
10 int main(void)
11 {
12         const char *s = "Hello";
13         struct Line *p, *q;
14         p = malloc(sizeof(*p));
15         p->string = strdup(s);  /* strdup makes a dynamic copy of s */
16         p->next = NULL;
```

- Avallokering sker i omvänd ordning. Viktigt att inte tappa bort någon pekare.

```
17         free(p->string);
18         q = p->next;
19         free(p);
20         p = q;
21         return 0;
22 }
```



Dynamisk allokering av `struct`

- Kombinationen dynamiskt minne, pekare och `struct` med pekar-fält är vanlig.

code/allocstruct.c

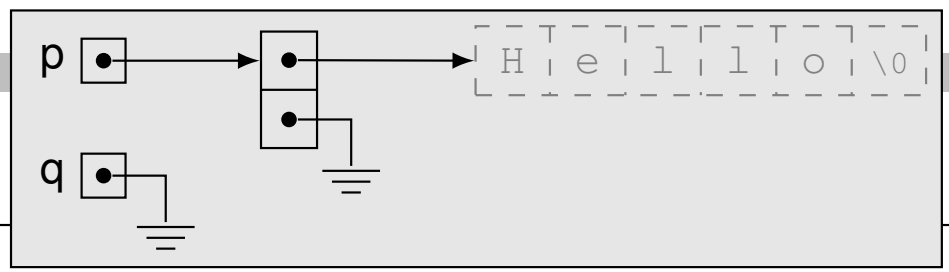
```
5  struct Line {
6          char *string;           /* Pointer to the actual string */
7          struct Line *next;      /* Pointer to next line */
8  };
```

- Vid konstruktion så måste minne allokeras, dels till `struct`en, dels till datat.

```
10 int main(void)
11 {
12         const char *s = "Hello";
13         struct Line *p, *q;
14         p = malloc(sizeof(*p));
15         p->string = strdup(s);  /* strdup makes a dynamic copy of s */
16         p->next = NULL;
```

- Avallokering sker i omvänd ordning. Viktigt att inte tappa bort någon pekare.

```
17         free(p->string);
18         q = p->next;
19         free(p);
20         p = q;
21         return 0;
22 }
```



Dynamisk allokering av `struct`

- Kombinationen dynamiskt minne, pekare och `struct` med pekar-fält är vanlig.

code/allocstruct.c

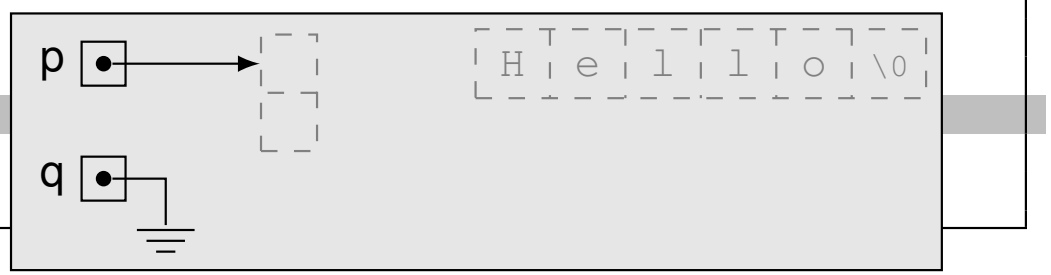
```
5  struct Line {
6          char *string;           /* Pointer to the actual string */
7          struct Line *next;      /* Pointer to next line */
8  };
```

- Vid konstruktion så måste minne allokeras, dels till `struct`en, dels till datat.

```
10 int main(void)
11 {
12         const char *s = "Hello";
13         struct Line *p, *q;
14         p = malloc(sizeof(*p));
15         p->string = strdup(s);  /* strdup makes a dynamic copy of s */
16         p->next = NULL;
```

- Avallokering sker i omvänd ordning. Viktigt att inte tappa bort någon pekare.

```
17         free(p->string);
18         q = p->next;
19         free(p);
20         p = q;
21         return 0;
22 }
```



Dynamisk allokering av `struct`

- Kombinationen dynamiskt minne, pekare och `struct` med pekar-fält är vanlig.

code/allocstruct.c

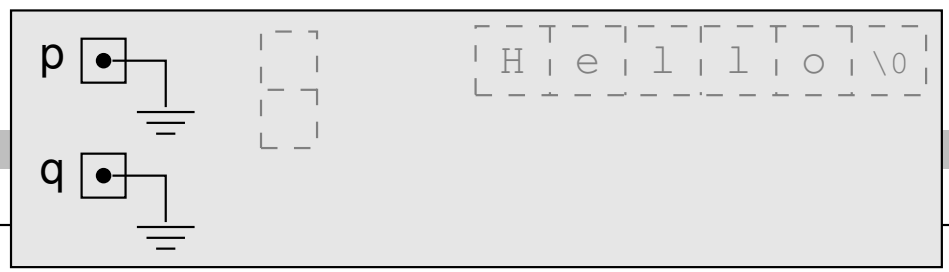
```
5  struct Line {
6          char *string;           /* Pointer to the actual string */
7          struct Line *next;      /* Pointer to next line */
8  };
```

- Vid konstruktion så måste minne allokeras, dels till `struct`en, dels till datat.

```
10 int main(void)
11 {
12         const char *s = "Hello";
13         struct Line *p, *q;
14         p = malloc(sizeof(*p));
15         p->string = strdup(s);  /* strdup makes a dynamic copy of s */
16         p->next = NULL;
```

- Avallokering sker i omvänd ordning. Viktigt att inte tappa bort någon pekare.

```
17         free(p->string);
18         q = p->next;
19         free(p);
20         p = q;
21         return 0;
22 }
```



Dynamisk allokering av `struct`

- Kombinationen dynamiskt minne, pekare och `struct` med pekar-fält är vanlig.

code/allocstruct.c

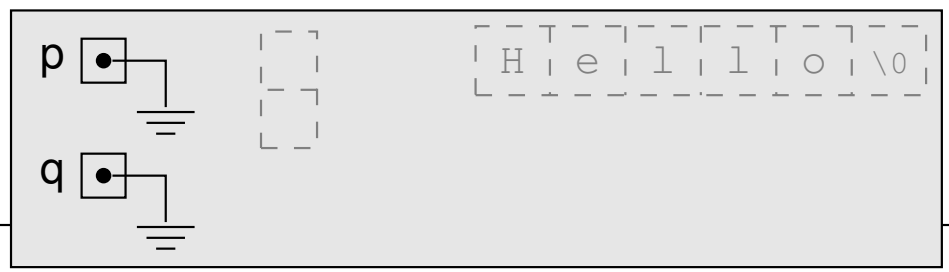
```
5  struct Line {
6          char *string;           /* Pointer to the actual string */
7          struct Line *next;      /* Pointer to next line */
8  };
```

- Vid konstruktion så måste minne allokeras, dels till `struct`en, dels till datat.

```
10 int main(void)
11 {
12         const char *s = "Hello";
13         struct Line *p, *q;
14         p = malloc(sizeof(*p));
15         p->string = strdup(s);  /* strdup makes a dynamic copy of s */
16         p->next = NULL;
```

- Avallokering sker i omvänd ordning. Viktigt att inte tappa bort någon pekare.

```
17         free(p->string);
18         q = p->next;
19         free(p);
20         p = q;
21         return 0;
22 }
```



Länkad lista i C

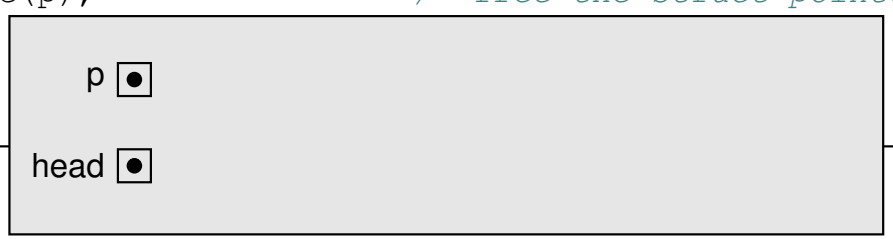
code/linkedlist.c

```c
struct Line {
        char *string;           /* Pointer to the actual string */
        struct Line *next;      /* Pointer to next line */
};

int main(void)
{
        const char *s = "First", *t = "Second";
        struct Line *p;                 /* work pointer */
        struct Line *head = NULL;       /* points to first line in list */
        /* Line 1 */
        p = malloc(sizeof(*p));         /* allocate struct #1 */
        p->string = strdup(s);          /* make a dynamic copy of s */
        p->next = head;                 /* set pointer to next line */
        head = p;                       /* set head to point to this line */
        /* Line 2 */
        p = malloc(sizeof(*p));         /* allocate struct #2 */
        p->string = strdup(t);          /* make a dynamic copy of t */
        p->next = head;                 /* set pointer to next line */
        head = p;                       /* set head to point to this line */
        /* Print content of list, in reverse */
        while (head != NULL) {          /* while we have remaining lines */
                p = head;               /* look at the first one... */
                printf("%s\n", p->string); /* print string */
                free(p->string);        /* free string */
                head = p->next;         /* grab next line */
                free(p);                /* free the struct pointer */
        }
        return 0;
}
```

Länkad lista i C

code/linkedlist.c

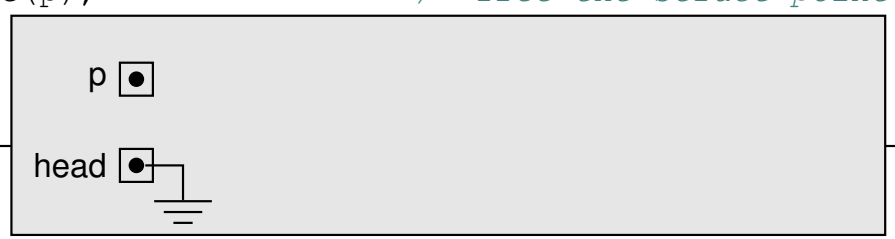
```c
struct Line {
        char *string;           /* Pointer to the actual string */
        struct Line *next;      /* Pointer to next line */
};

int main(void)
{
        const char *s = "First", *t = "Second";
        struct Line *p;                 /* work pointer */
        struct Line *head = NULL;       /* points to first line in list */
        /* Line 1 */
        p = malloc(sizeof(*p));         /* allocate struct #1 */
        p->string = strdup(s);          /* make a dynamic copy of s */
        p->next = head;                 /* set pointer to next line */
        head = p;                       /* set head to point to this line */
        /* Line 2 */
        p = malloc(sizeof(*p));         /* allocate struct #2 */
        p->string = strdup(t);          /* make a dynamic copy of t */
        p->next = head;                 /* set pointer to next line */
        head = p;                       /* set head to point to this line */
        /* Print content of list, in reverse */
        while (head != NULL) {          /* while we have remaining lines */
                p = head;               /* look at the first one... */
                printf("%s\n", p->string); /* print string */
                free(p->string);        /* free string */
                head = p->next;         /* grab next line */
                free(p);                /* free the struct pointer */
        }
        return 0;
}
```



Länkad lista i C

code/linkedlist.c

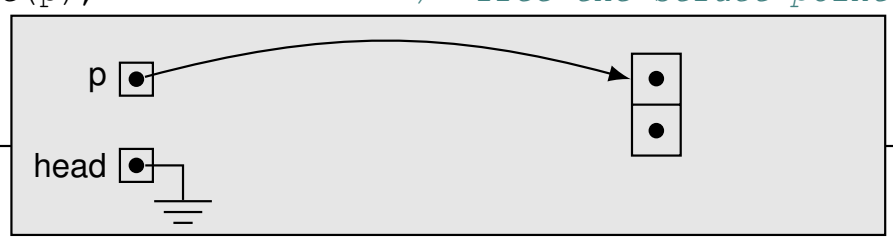
```
5   struct Line {
6           char *string;           /* Pointer to the actual string */
7           struct Line *next;      /* Pointer to next line */
8   };
9
10  int main(void)
11  {
12          const char *s = "First", *t = "Second";
13          struct Line *p;                 /* work pointer */
14          struct Line *head = NULL;       /* points to first line in list */
15          /* Line 1 */
16          p = malloc(sizeof(*p));         /* allocate struct #1 */
17          p->string = strdup(s);          /* make a dynamic copy of s */
18          p->next = head;                 /* set pointer to next line */
19          head = p;                       /* set head to point to this line */
20          /* Line 2 */
21          p = malloc(sizeof(*p));         /* allocate struct #2 */
22          p->string = strdup(t);          /* make a dynamic copy of t */
23          p->next = head;                 /* set pointer to next line */
24          head = p;                       /* set head to point to this line */
25          /* Print content of list, in reverse */
26          while (head != NULL) {          /* while we have remaining lines */
27                  p = head;               /* look at the first one... */
28                  printf("%s\n", p->string); /* print string */
29                  free(p->string);        /* free string */
30                  head = p->next;         /* grab next line */
31                  free(p);                /* free the struct pointer */
32          }
33          return 0;
34  }
```



Länkad lista i C

code/linkedlist.c

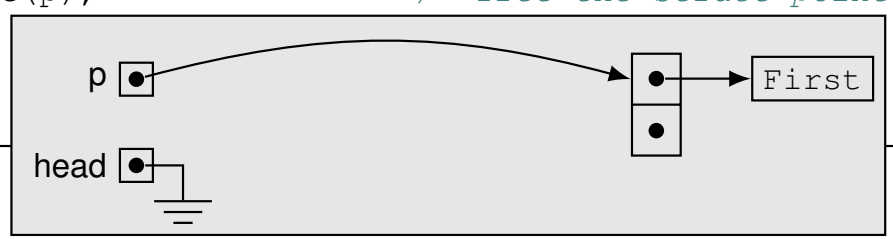
```c
struct Line {
        char *string;           /* Pointer to the actual string */
        struct Line *next;      /* Pointer to next line */
};

int main(void)
{
        const char *s = "First", *t = "Second";
        struct Line *p;                 /* work pointer */
        struct Line *head = NULL;       /* points to first line in list */
        /* Line 1 */
        p = malloc(sizeof(*p));         /* allocate struct #1 */
        p->string = strdup(s);          /* make a dynamic copy of s */
        p->next = head;                 /* set pointer to next line */
        head = p;                       /* set head to point to this line */
        /* Line 2 */
        p = malloc(sizeof(*p));         /* allocate struct #2 */
        p->string = strdup(t);          /* make a dynamic copy of t */
        p->next = head;                 /* set pointer to next line */
        head = p;                       /* set head to point to this line */
        /* Print content of list, in reverse */
        while (head != NULL) {          /* while we have remaining lines */
                p = head;               /* look at the first one... */
                printf("%s\n", p->string); /* print string */
                free(p->string);        /* free string */
                head = p->next;         /* grab next line */
                free(p);                /* free the struct pointer */
        }
        return 0;
}
```



Länkad lista i C

code/linkedlist.c

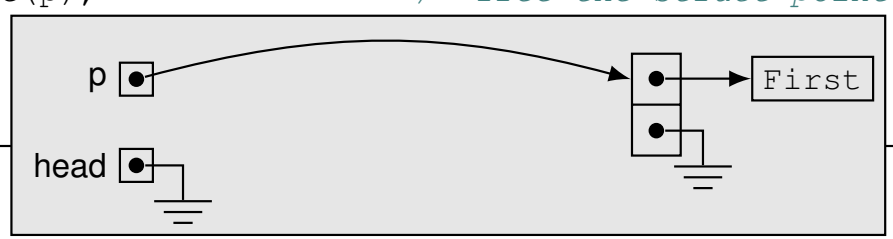
```c
struct Line {
        char *string;           /* Pointer to the actual string */
        struct Line *next;      /* Pointer to next line */
};

int main(void)
{
        const char *s = "First", *t = "Second";
        struct Line *p;                 /* work pointer */
        struct Line *head = NULL;       /* points to first line in list */
        /* Line 1 */
        p = malloc(sizeof(*p));         /* allocate struct #1 */
        p->string = strdup(s);          /* make a dynamic copy of s */
        p->next = head;                 /* set pointer to next line */
        head = p;                       /* set head to point to this line */
        /* Line 2 */
        p = malloc(sizeof(*p));         /* allocate struct #2 */
        p->string = strdup(t);          /* make a dynamic copy of t */
        p->next = head;                 /* set pointer to next line */
        head = p;                       /* set head to point to this line */
        /* Print content of list, in reverse */
        while (head != NULL) {          /* while we have remaining lines */
                p = head;               /* look at the first one... */
                printf("%s\n", p->string); /* print string */
                free(p->string);        /* free string */
                head = p->next;         /* grab next line */
                free(p);                /* free the struct pointer */
        }
        return 0;
}
```



Länkad lista i C

code/linkedlist.c

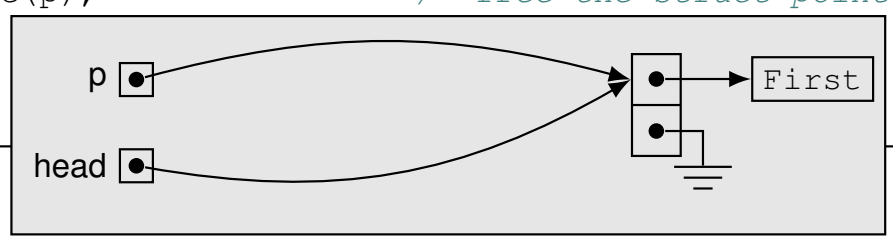
```c
struct Line {
        char *string;           /* Pointer to the actual string */
        struct Line *next;      /* Pointer to next line */
};

int main(void)
{
        const char *s = "First", *t = "Second";
        struct Line *p;                 /* work pointer */
        struct Line *head = NULL;       /* points to first line in list */
        /* Line 1 */
        p = malloc(sizeof(*p));         /* allocate struct #1 */
        p->string = strdup(s);          /* make a dynamic copy of s */
        p->next = head;                 /* set pointer to next line */
        head = p;                       /* set head to point to this line */
        /* Line 2 */
        p = malloc(sizeof(*p));         /* allocate struct #2 */
        p->string = strdup(t);          /* make a dynamic copy of t */
        p->next = head;                 /* set pointer to next line */
        head = p;                       /* set head to point to this line */
        /* Print content of list, in reverse */
        while (head != NULL) {          /* while we have remaining lines */
                p = head;               /* look at the first one... */
                printf("%s\n", p->string); /* print string */
                free(p->string);        /* free string */
                head = p->next;         /* grab next line */
                free(p);                /* free the struct pointer */
        }
        return 0;
}
```



Länkad lista i C

code/linkedlist.c

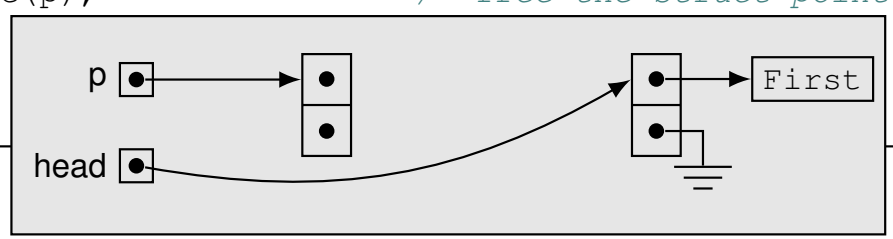
```c
struct Line {
        char *string;           /* Pointer to the actual string */
        struct Line *next;      /* Pointer to next line */
};

int main(void)
{
        const char *s = "First", *t = "Second";
        struct Line *p;                 /* work pointer */
        struct Line *head = NULL;       /* points to first line in list */
        /* Line 1 */
        p = malloc(sizeof(*p));         /* allocate struct #1 */
        p->string = strdup(s);          /* make a dynamic copy of s */
        p->next = head;                 /* set pointer to next line */
        head = p;                       /* set head to point to this line */
        /* Line 2 */
        p = malloc(sizeof(*p));         /* allocate struct #2 */
        p->string = strdup(t);          /* make a dynamic copy of t */
        p->next = head;                 /* set pointer to next line */
        head = p;                       /* set head to point to this line */
        /* Print content of list, in reverse */
        while (head != NULL) {          /* while we have remaining lines */
                p = head;               /* look at the first one... */
                printf("%s\n", p->string); /* print string */
                free(p->string);        /* free string */
                head = p->next;         /* grab next line */
                free(p);                /* free the struct pointer */
        }
        return 0;
}
```



Länkad lista i C

code/linkedlist.c

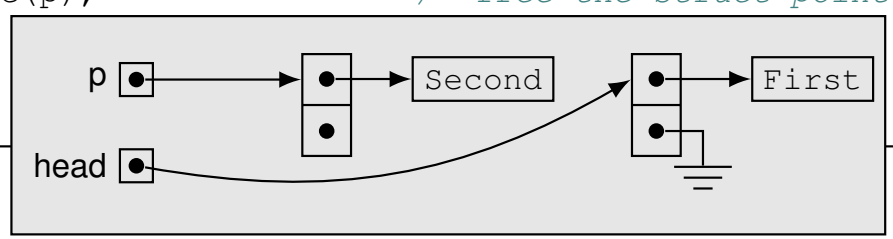
```c
struct Line {
        char *string;           /* Pointer to the actual string */
        struct Line *next;      /* Pointer to next line */
};

int main(void)
{
        const char *s = "First", *t = "Second";
        struct Line *p;                 /* work pointer */
        struct Line *head = NULL;       /* points to first line in list */
        /* Line 1 */
        p = malloc(sizeof(*p));         /* allocate struct #1 */
        p->string = strdup(s);          /* make a dynamic copy of s */
        p->next = head;                 /* set pointer to next line */
        head = p;                       /* set head to point to this line */
        /* Line 2 */
        p = malloc(sizeof(*p));         /* allocate struct #2 */
        p->string = strdup(t);          /* make a dynamic copy of t */
        p->next = head;                 /* set pointer to next line */
        head = p;                       /* set head to point to this line */
        /* Print content of list, in reverse */
        while (head != NULL) {          /* while we have remaining lines */
                p = head;               /* look at the first one... */
                printf("%s\n", p->string); /* print string */
                free(p->string);        /* free string */
                head = p->next;         /* grab next line */
                free(p);                /* free the struct pointer */
        }
        return 0;
}
```



Länkad lista i C

code/linkedlist.c

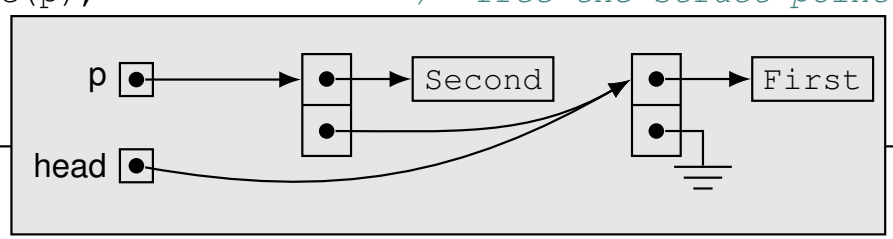
```
5   struct Line {
6           char *string;           /* Pointer to the actual string */
7           struct Line *next;      /* Pointer to next line */
8   };
9
10  int main(void)
11  {
12          const char *s = "First", *t = "Second";
13          struct Line *p;                 /* work pointer */
14          struct Line *head = NULL;       /* points to first line in list */
15          /* Line 1 */
16          p = malloc(sizeof(*p));         /* allocate struct #1 */
17          p->string = strdup(s);          /* make a dynamic copy of s */
18          p->next = head;                 /* set pointer to next line */
19          head = p;                       /* set head to point to this line */
20          /* Line 2 */
21          p = malloc(sizeof(*p));         /* allocate struct #2 */
22          p->string = strdup(t);          /* make a dynamic copy of t */
23          p->next = head;                 /* set pointer to next line */
24          head = p;                       /* set head to point to this line */
25          /* Print content of list, in reverse */
26          while (head != NULL) {          /* while we have remaining lines */
27                  p = head;               /* look at the first one... */
28                  printf("%s\n", p->string); /* print string */
29                  free(p->string);        /* free string */
30                  head = p->next;         /* grab next line */
31                  free(p);                /* free the struct pointer */
32          }
33          return 0;
34  }
```



Länkad lista i C

code/linkedlist.c

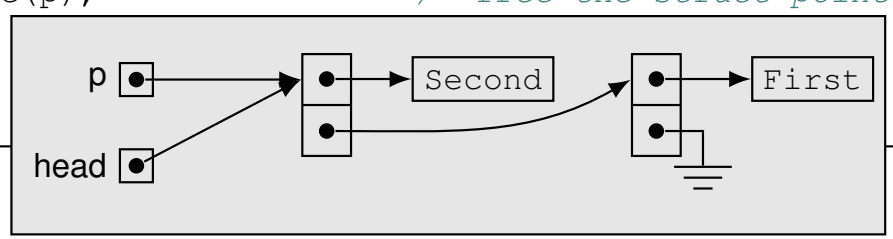
```c
struct Line {
        char *string;           /* Pointer to the actual string */
        struct Line *next;      /* Pointer to next line */
};

int main(void)
{
        const char *s = "First", *t = "Second";
        struct Line *p;                 /* work pointer */
        struct Line *head = NULL;       /* points to first line in list */
        /* Line 1 */
        p = malloc(sizeof(*p));         /* allocate struct #1 */
        p->string = strdup(s);          /* make a dynamic copy of s */
        p->next = head;                 /* set pointer to next line */
        head = p;                       /* set head to point to this line */
        /* Line 2 */
        p = malloc(sizeof(*p));         /* allocate struct #2 */
        p->string = strdup(t);          /* make a dynamic copy of t */
        p->next = head;                 /* set pointer to next line */
        head = p;                       /* set head to point to this line */
        /* Print content of list, in reverse */
        while (head != NULL) {          /* while we have remaining lines */
                p = head;               /* look at the first one... */
                printf("%s\n", p->string); /* print string */
                free(p->string);        /* free string */
                head = p->next;         /* grab next line */
                free(p);                /* free the struct pointer */
        }
        return 0;
}
```



Länkad lista i C

code/linkedlist.c

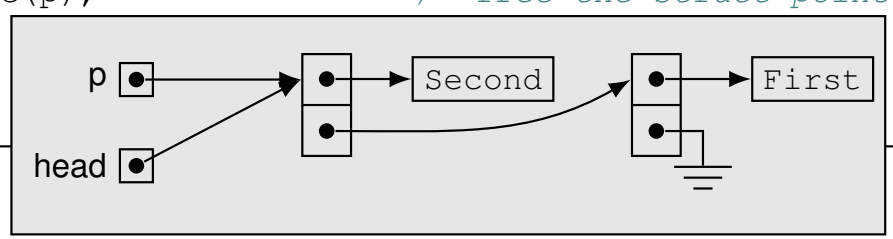
```c
struct Line {
        char *string;           /* Pointer to the actual string */
        struct Line *next;      /* Pointer to next line */
};

int main(void)
{
        const char *s = "First", *t = "Second";
        struct Line *p;                 /* work pointer */
        struct Line *head = NULL;       /* points to first line in list */
        /* Line 1 */
        p = malloc(sizeof(*p));         /* allocate struct #1 */
        p->string = strdup(s);          /* make a dynamic copy of s */
        p->next = head;                 /* set pointer to next line */
        head = p;                       /* set head to point to this line */
        /* Line 2 */
        p = malloc(sizeof(*p));         /* allocate struct #2 */
        p->string = strdup(t);          /* make a dynamic copy of t */
        p->next = head;                 /* set pointer to next line */
        head = p;                       /* set head to point to this line */
        /* Print content of list, in reverse */
        while (head != NULL) {          /* while we have remaining lines */
                p = head;               /* look at the first one... */
                printf("%s\n", p->string); /* print string */
                free(p->string);        /* free string */
                head = p->next;         /* grab next line */
                free(p);                /* free the struct pointer */
        }
        return 0;
}
```



Länkad lista i C

code/linkedlist.c

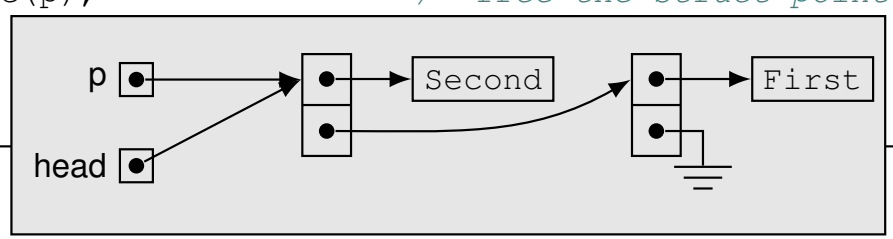
```
5   struct Line {
6           char *string;           /* Pointer to the actual string */
7           struct Line *next;      /* Pointer to next line */
8   };
9
10  int main(void)
11  {
12          const char *s = "First", *t = "Second";
13          struct Line *p;                 /* work pointer */
14          struct Line *head = NULL;       /* points to first line in list */
15          /* Line 1 */
16          p = malloc(sizeof(*p));         /* allocate struct #1 */
17          p->string = strdup(s);          /* make a dynamic copy of s */
18          p->next = head;                 /* set pointer to next line */
19          head = p;                       /* set head to point to this line */
20          /* Line 2 */
21          p = malloc(sizeof(*p));         /* allocate struct #2 */
22          p->string = strdup(t);          /* make a dynamic copy of t */
23          p->next = head;                 /* set pointer to next line */
24          head = p;                       /* set head to point to this line */
25          /* Print content of list, in reverse */
26          while (head != NULL) {          /* while we have remaining lines */
27                  p = head;               /* look at the first one... */
28                  printf("%s\n", p->string); /* print string */
29                  free(p->string);        /* free string */
30                  head = p->next;         /* grab next line */
31                  free(p);                /* free the struct pointer */
32          }
33          return 0;
34  }
```



Länkad lista i C

code/linkedlist.c

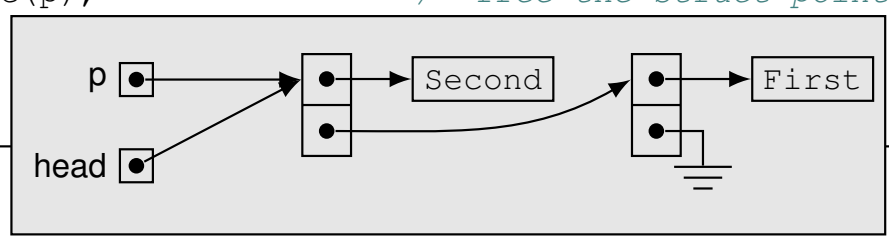
```
5   struct Line {
6           char *string;           /* Pointer to the actual string */
7           struct Line *next;      /* Pointer to next line */
8   };
9
10  int main(void)
11  {
12          const char *s = "First", *t = "Second";
13          struct Line *p;                 /* work pointer */
14          struct Line *head = NULL;       /* points to first line in list */
15          /* Line 1 */
16          p = malloc(sizeof(*p));         /* allocate struct #1 */
17          p->string = strdup(s);          /* make a dynamic copy of s */
18          p->next = head;                 /* set pointer to next line */
19          head = p;                       /* set head to point to this line */
20          /* Line 2 */
21          p = malloc(sizeof(*p));         /* allocate struct #2 */
22          p->string = strdup(t);          /* make a dynamic copy of t */
23          p->next = head;                 /* set pointer to next line */
24          head = p;                       /* set head to point to this line */
25          /* Print content of list, in reverse */
26          while (head != NULL) {          /* while we have remaining lines */
27                  p = head;               /* look at the first one... */
28                  printf("%s\n", p->string); /* print string */
29                  free(p->string);        /* free string */
30                  head = p->next;         /* grab next line */
31                  free(p);                /* free the struct pointer */
32          }
33          return 0;
34  }
```



Länkad lista i C

code/linkedlist.c

```
5   struct Line {
6           char *string;           /* Pointer to the actual string */
7           struct Line *next;      /* Pointer to next line */
8   };
9
10  int main(void)
11  {
12          const char *s = "First", *t = "Second";
13          struct Line *p;                 /* work pointer */
14          struct Line *head = NULL;       /* points to first line in list */
15          /* Line 1 */
16          p = malloc(sizeof(*p));         /* allocate struct #1 */
17          p->string = strdup(s);          /* make a dynamic copy of s */
18          p->next = head;                 /* set pointer to next line */
19          head = p;                       /* set head to point to this line */
20          /* Line 2 */
21          p = malloc(sizeof(*p));         /* allocate struct #2 */
22          p->string = strdup(t);          /* make a dynamic copy of t */
23          p->next = head;                 /* set pointer to next line */
24          head = p;                       /* set head to point to this line */
25          /* Print content of list, in reverse */
26          while (head != NULL) {          /* while we have remaining lines */
27                  p = head;               /* look at the first one... */
28                  printf("%s\n", p->string); /* print string */
29                  free(p->string);        /* free string */
30                  head = p->next;         /* grab next line */
31                  free(p);                /* free the struct pointer */
32          }
33          return 0;
34  }
```

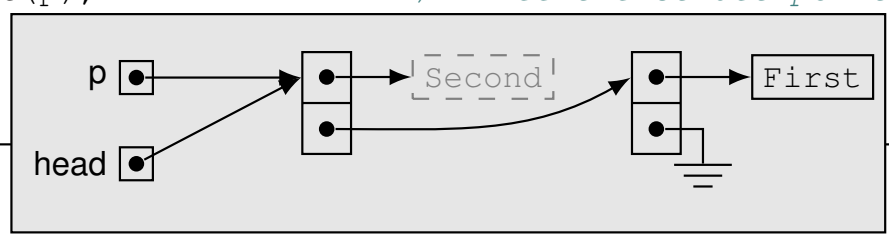


Second

Länkad lista i C

code/linkedlist.c

```
5   struct Line {
6           char *string;           /* Pointer to the actual string */
7           struct Line *next;      /* Pointer to next line */
8   };
9
10  int main(void)
11  {
12          const char *s = "First", *t = "Second";
13          struct Line *p;                 /* work pointer */
14          struct Line *head = NULL;       /* points to first line in list */
15          /* Line 1 */
16          p = malloc(sizeof(*p));         /* allocate struct #1 */
17          p->string = strdup(s);          /* make a dynamic copy of s */
18          p->next = head;                 /* set pointer to next line */
19          head = p;                       /* set head to point to this line */
20          /* Line 2 */
21          p = malloc(sizeof(*p));         /* allocate struct #2 */
22          p->string = strdup(t);          /* make a dynamic copy of t */
23          p->next = head;                 /* set pointer to next line */
24          head = p;                       /* set head to point to this line */
25          /* Print content of list, in reverse */
26          while (head != NULL) {          /* while we have remaining lines */
27                  p = head;               /* look at the first one... */
28                  printf("%s\n", p->string); /* print string */
29                  free(p->string);        /* free string */
30                  head = p->next;         /* grab next line */
31                  free(p);                /* free the struct pointer */
32          }
33          return 0;
34  }
```

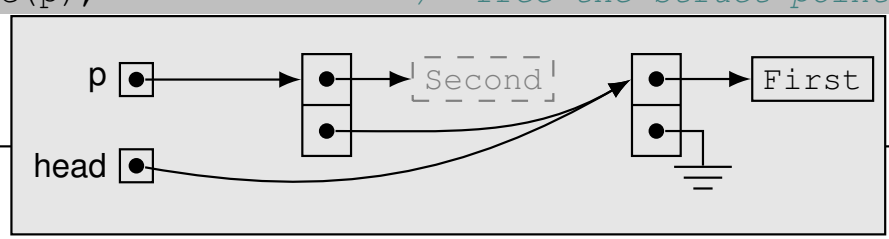


Second

Länkad lista i C

code/linkedlist.c

```
5   struct Line {
6           char *string;           /* Pointer to the actual string */
7           struct Line *next;      /* Pointer to next line */
8   };
9
10  int main(void)
11  {
12          const char *s = "First", *t = "Second";
13          struct Line *p;                 /* work pointer */
14          struct Line *head = NULL;       /* points to first line in list */
15          /* Line 1 */
16          p = malloc(sizeof(*p));         /* allocate struct #1 */
17          p->string = strdup(s);          /* make a dynamic copy of s */
18          p->next = head;                 /* set pointer to next line */
19          head = p;                       /* set head to point to this line */
20          /* Line 2 */
21          p = malloc(sizeof(*p));         /* allocate struct #2 */
22          p->string = strdup(t);          /* make a dynamic copy of t */
23          p->next = head;                 /* set pointer to next line */
24          head = p;                       /* set head to point to this line */
25          /* Print content of list, in reverse */
26          while (head != NULL) {          /* while we have remaining lines */
27                  p = head;               /* look at the first one... */
28                  printf("%s\n", p->string); /* print string */
29                  free(p->string);        /* free string */
30                  head = p->next;         /* grab next line */
31                  free(p);                /* free the struct pointer */
32          }
33          return 0;
34  }
```

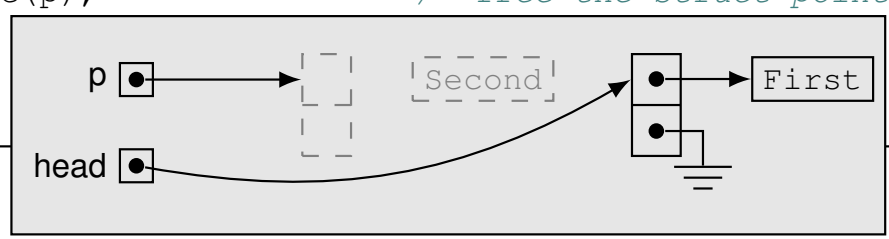


Second

Länkad lista i C

code/linkedlist.c

```c
struct Line {
        char *string;           /* Pointer to the actual string */
        struct Line *next;      /* Pointer to next line */
};

int main(void)
{
        const char *s = "First", *t = "Second";
        struct Line *p;                 /* work pointer */
        struct Line *head = NULL;       /* points to first line in list */
        /* Line 1 */
        p = malloc(sizeof(*p));         /* allocate struct #1 */
        p->string = strdup(s);          /* make a dynamic copy of s */
        p->next = head;                 /* set pointer to next line */
        head = p;                       /* set head to point to this line */
        /* Line 2 */
        p = malloc(sizeof(*p));         /* allocate struct #2 */
        p->string = strdup(t);          /* make a dynamic copy of t */
        p->next = head;                 /* set pointer to next line */
        head = p;                       /* set head to point to this line */
        /* Print content of list, in reverse */
        while (head != NULL) {          /* while we have remaining lines */
                p = head;               /* look at the first one... */
                printf("%s\n", p->string); /* print string */
                free(p->string);        /* free string */
                head = p->next;         /* grab next line */
                free(p);                /* free the struct pointer */
        }
        return 0;
}
```

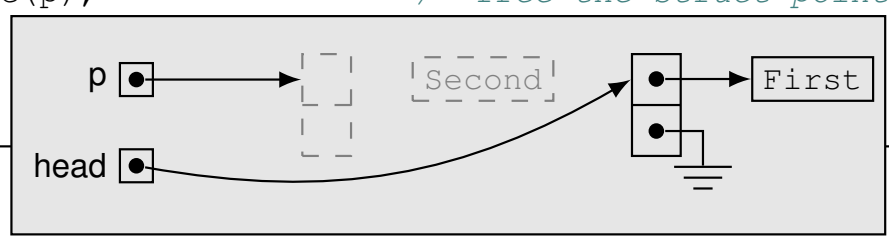


Second

Länkad lista i C

code/linkedlist.c

```c
struct Line {
        char *string;           /* Pointer to the actual string */
        struct Line *next;      /* Pointer to next line */
};

int main(void)
{
        const char *s = "First", *t = "Second";
        struct Line *p;                 /* work pointer */
        struct Line *head = NULL;       /* points to first line in list */
        /* Line 1 */
        p = malloc(sizeof(*p));         /* allocate struct #1 */
        p->string = strdup(s);          /* make a dynamic copy of s */
        p->next = head;                 /* set pointer to next line */
        head = p;                       /* set head to point to this line */
        /* Line 2 */
        p = malloc(sizeof(*p));         /* allocate struct #2 */
        p->string = strdup(t);          /* make a dynamic copy of t */
        p->next = head;                 /* set pointer to next line */
        head = p;                       /* set head to point to this line */
        /* Print content of list, in reverse */
        while (head != NULL) {          /* while we have remaining lines */
                p = head;               /* look at the first one... */
                printf("%s\n", p->string); /* print string */
                free(p->string);        /* free string */
                head = p->next;         /* grab next line */
                free(p);                /* free the struct pointer */
        }
        return 0;
}
```

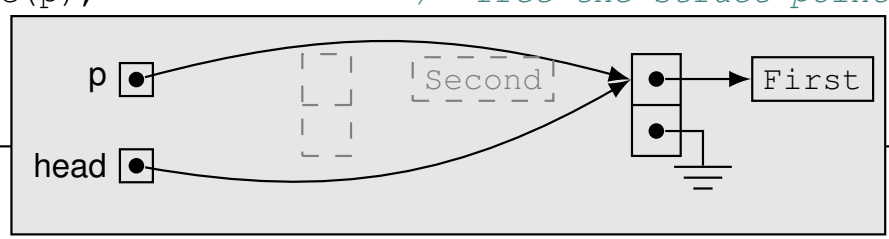


Second

Länkad lista i C

code/linkedlist.c

```
5   struct Line {
6           char *string;           /* Pointer to the actual string */
7           struct Line *next;      /* Pointer to next line */
8   };
9
10  int main(void)
11  {
12          const char *s = "First", *t = "Second";
13          struct Line *p;                 /* work pointer */
14          struct Line *head = NULL;       /* points to first line in list */
15          /* Line 1 */
16          p = malloc(sizeof(*p));         /* allocate struct #1 */
17          p->string = strdup(s);          /* make a dynamic copy of s */
18          p->next = head;                 /* set pointer to next line */
19          head = p;                       /* set head to point to this line */
20          /* Line 2 */
21          p = malloc(sizeof(*p));         /* allocate struct #2 */
22          p->string = strdup(t);          /* make a dynamic copy of t */
23          p->next = head;                 /* set pointer to next line */
24          head = p;                       /* set head to point to this line */
25          /* Print content of list, in reverse */
26          while (head != NULL) {          /* while we have remaining lines */
27                  p = head;               /* look at the first one... */
28                  printf("%s\n", p->string); /* print string */
29                  free(p->string);        /* free string */
30                  head = p->next;         /* grab next line */
31                  free(p);                /* free the struct pointer */
32          }
33          return 0;
34  }
```

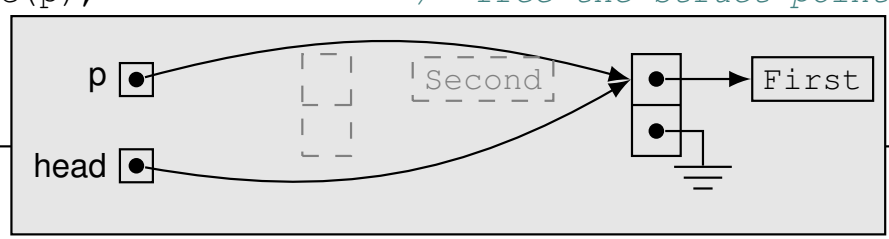


Second

Länkad lista i C

code/linkedlist.c

```
5   struct Line {
6           char *string;           /* Pointer to the actual string */
7           struct Line *next;      /* Pointer to next line */
8   };
9
10  int main(void)
11  {
12          const char *s = "First", *t = "Second";
13          struct Line *p;                 /* work pointer */
14          struct Line *head = NULL;       /* points to first line in list */
15          /* Line 1 */
16          p = malloc(sizeof(*p));         /* allocate struct #1 */
17          p->string = strdup(s);          /* make a dynamic copy of s */
18          p->next = head;                 /* set pointer to next line */
19          head = p;                       /* set head to point to this line */
20          /* Line 2 */
21          p = malloc(sizeof(*p));         /* allocate struct #2 */
22          p->string = strdup(t);          /* make a dynamic copy of t */
23          p->next = head;                 /* set pointer to next line */
24          head = p;                       /* set head to point to this line */
25          /* Print content of list, in reverse */
26          while (head != NULL) {          /* while we have remaining lines */
27                  p = head;               /* look at the first one... */
28                  printf("%s\n", p->string); /* print string */
29                  free(p->string);        /* free string */
30                  head = p->next;         /* grab next line */
31                  free(p);                /* free the struct pointer */
32          }
33          return 0;
34  }
```

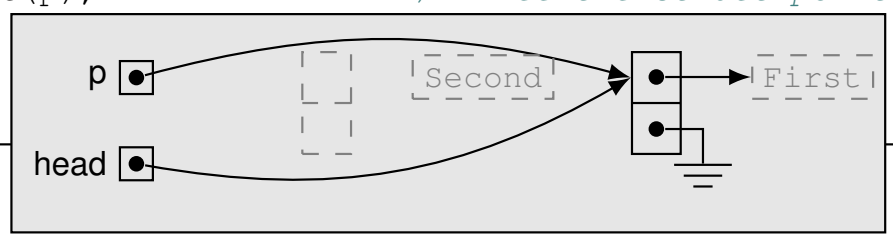



```
Second
First
```

Länkad lista i C

code/linkedlist.c

```
5   struct Line {
6           char *string;           /* Pointer to the actual string */
7           struct Line *next;      /* Pointer to next line */
8   };
9
10  int main(void)
11  {
12          const char *s = "First", *t = "Second";
13          struct Line *p;                 /* work pointer */
14          struct Line *head = NULL;       /* points to first line in list */
15          /* Line 1 */
16          p = malloc(sizeof(*p));         /* allocate struct #1 */
17          p->string = strdup(s);          /* make a dynamic copy of s */
18          p->next = head;                 /* set pointer to next line */
19          head = p;                       /* set head to point to this line */
20          /* Line 2 */
21          p = malloc(sizeof(*p));         /* allocate struct #2 */
22          p->string = strdup(t);          /* make a dynamic copy of t */
23          p->next = head;                 /* set pointer to next line */
24          head = p;                       /* set head to point to this line */
25          /* Print content of list, in reverse */
26          while (head != NULL) {          /* while we have remaining lines */
27                  p = head;               /* look at the first one... */
28                  printf("%s\n", p->string); /* print string */
29                  free(p->string);        /* free string */
30                  head = p->next;         /* grab next line */
31                  free(p);                /* free the struct pointer */
32          }
33          return 0;
34  }
```

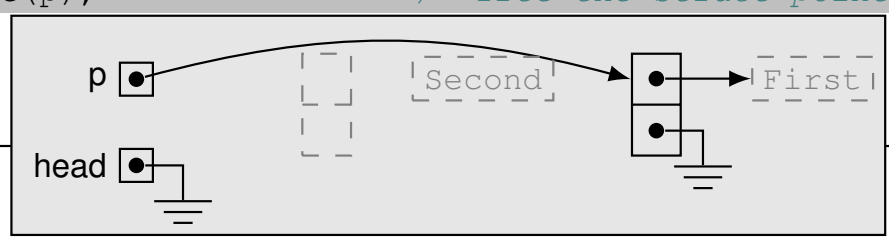



```
Second
First
```

Länkad lista i C

code/linkedlist.c

```
5   struct Line {
6           char *string;           /* Pointer to the actual string */
7           struct Line *next;      /* Pointer to next line */
8   };
9
10  int main(void)
11  {
12          const char *s = "First", *t = "Second";
13          struct Line *p;                 /* work pointer */
14          struct Line *head = NULL;       /* points to first line in list */
15          /* Line 1 */
16          p = malloc(sizeof(*p));         /* allocate struct #1 */
17          p->string = strdup(s);          /* make a dynamic copy of s */
18          p->next = head;                 /* set pointer to next line */
19          head = p;                       /* set head to point to this line */
20          /* Line 2 */
21          p = malloc(sizeof(*p));         /* allocate struct #2 */
22          p->string = strdup(t);          /* make a dynamic copy of t */
23          p->next = head;                 /* set pointer to next line */
24          head = p;                       /* set head to point to this line */
25          /* Print content of list, in reverse */
26          while (head != NULL) {          /* while we have remaining lines */
27                  p = head;               /* look at the first one... */
28                  printf("%s\n", p->string); /* print string */
29                  free(p->string);        /* free string */
30                  head = p->next;         /* grab next line */
31                  free(p);                /* free the struct pointer */
32          }
33          return 0;
34  }
```

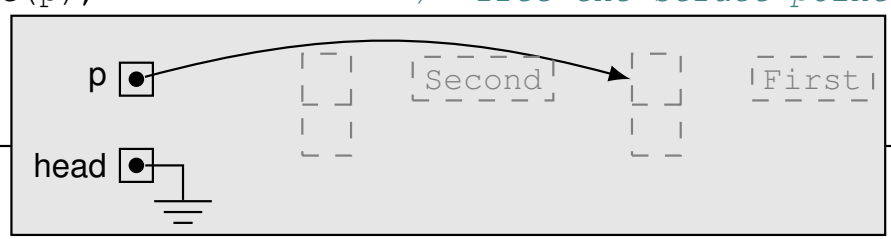



```
Second
First
```

Länkad lista i C

code/linkedlist.c

```c
struct Line {
        char *string;           /* Pointer to the actual string */
        struct Line *next;      /* Pointer to next line */
};

int main(void)
{
        const char *s = "First", *t = "Second";
        struct Line *p;                 /* work pointer */
        struct Line *head = NULL;       /* points to first line in list */
        /* Line 1 */
        p = malloc(sizeof(*p));         /* allocate struct #1 */
        p->string = strdup(s);          /* make a dynamic copy of s */
        p->next = head;                 /* set pointer to next line */
        head = p;                       /* set head to point to this line */
        /* Line 2 */
        p = malloc(sizeof(*p));         /* allocate struct #2 */
        p->string = strdup(t);          /* make a dynamic copy of t */
        p->next = head;                 /* set pointer to next line */
        head = p;                       /* set head to point to this line */
        /* Print content of list, in reverse */
        while (head != NULL) {          /* while we have remaining lines */
                p = head;               /* look at the first one... */
                printf("%s\n", p->string); /* print string */
                free(p->string);        /* free string */
                head = p->next;         /* grab next line */
                free(p);                /* free the struct pointer */
        }
        return 0;
}
```

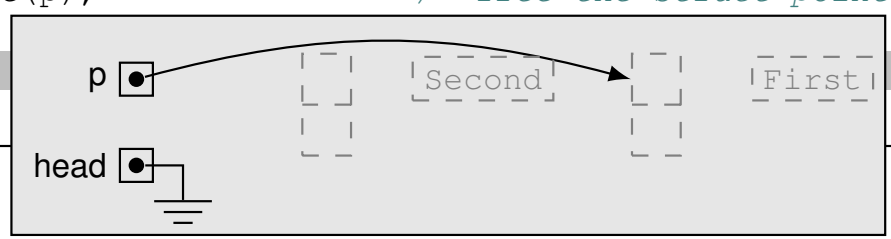



```
Second
First
```

Länkad lista i C

code/linkedlist.c

```c
struct Line {
        char *string;           /* Pointer to the actual string */
        struct Line *next;      /* Pointer to next line */
};

int main(void)
{
        const char *s = "First", *t = "Second";
        struct Line *p;                 /* work pointer */
        struct Line *head = NULL;       /* points to first line in list */
        /* Line 1 */
        p = malloc(sizeof(*p));         /* allocate struct #1 */
        p->string = strdup(s);          /* make a dynamic copy of s */
        p->next = head;                 /* set pointer to next line */
        head = p;                       /* set head to point to this line */
        /* Line 2 */
        p = malloc(sizeof(*p));         /* allocate struct #2 */
        p->string = strdup(t);          /* make a dynamic copy of t */
        p->next = head;                 /* set pointer to next line */
        head = p;                       /* set head to point to this line */
        /* Print content of list, in reverse */
        while (head != NULL) {          /* while we have remaining lines */
                p = head;               /* look at the first one... */
                printf("%s\n", p->string); /* print string */
                free(p->string);        /* free string */
                head = p->next;         /* grab next line */
                free(p);                /* free the struct pointer */
        }
        return 0;
}
```

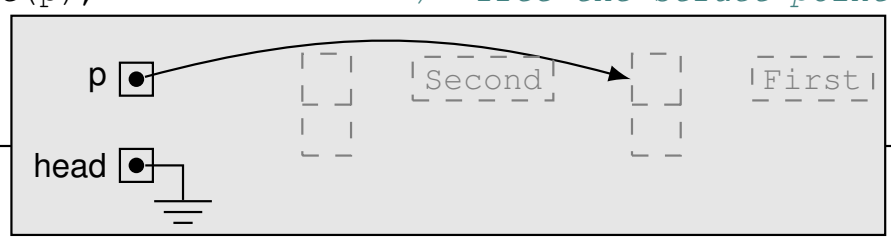



```
Second
First
```

Länkad lista i C

code/linkedlist.c

```c
struct Line {
        char *string;           /* Pointer to the actual string */
        struct Line *next;      /* Pointer to next line */
};

int main(void)
{
        const char *s = "First", *t = "Second";
        struct Line *p;                 /* work pointer */
        struct Line *head = NULL;       /* points to first line in list */
        /* Line 1 */
        p = malloc(sizeof(*p));         /* allocate struct #1 */
        p->string = strdup(s);          /* make a dynamic copy of s */
        p->next = head;                 /* set pointer to next line */
        head = p;                       /* set head to point to this line */
        /* Line 2 */
        p = malloc(sizeof(*p));         /* allocate struct #2 */
        p->string = strdup(t);          /* make a dynamic copy of t */
        p->next = head;                 /* set pointer to next line */
        head = p;                       /* set head to point to this line */
        /* Print content of list, in reverse */
        while (head != NULL) {          /* while we have remaining lines */
                p = head;               /* look at the first one... */
                printf("%s\n", p->string); /* print string */
                free(p->string);        /* free string */
                head = p->next;         /* grab next line */
                free(p);                /* free the struct pointer */
        }
        return 0;
}
```




```
Second
First
```