ΗΡΥ 201– Ψηφιακοί Υπολογιστές

CLANG: Chania C-Level Assembly Language (ή «Intermediate Code»)

Μοντέλο προγραμματισμού MIPS

- Οι λέξεις (words) έχουν πλάτος 32 bits
- 32 καταχωρητές
- Ο κάθε καταχωρητής έχει πλάτος 32 bits
- Η πρόσβαση στη μνήμη γίνεται με χρήση των εντολών load, store.
- Όλες οι αριθμητικές και λογικές πράξεις αποθηκεύουν το αποτέλεσμά τους σε καταχωρητή
- Η εκτέλεση των εντολών γίνεται στην σειρά, εκτός εάν η ροή του προγράμματος αλλάξει λόγω εντολής ελέγχου ροής

CLANG

- Η CLANG είναι μία ενδιάμεση “γλώσσα” προγραμματισμού
- Στόχος μαθήματος είναι η εξεικοίωση με την γλώσσα μηχανής (assembly MIPS)
- C → CLANG → Assembly MIPS

Δήλωση Μεταβλητών και Συναρτήσεων

- Οι δηλώσεις και οι αρχικοποιήσεις των μεταβλητών καθώς και οι δηλώσεις των συναρτήσεων γίνονται όπως ακριβώς στην C.
- **ΠΡΟΣΟΧΗ!** Η κάθε μεταβλητή αποτελεί χώρο που έχει δεσμευτεί στην μνήμη.
- Παράδειγμα:

C	CLANG

int main()
{
int x = 5;
……
}
int foo()
{
……
}
	Int R0 = 0, R1, R2, R3, R4, ….
	int main()
	{
	int x = 5;
	……
	}
	int foo()
	{
	……
	}

Επεξεργασία (I)

- Οι υπολογισμοί δεν γίνονται με την βοήθεια των μεταβλητών αλλά μόνο των καταχωρητών.
- **ΠΡΟΣΟΧΗ!** Πριν την επεξεργασία μίας μεταβλητής θα πρέπει πρώτα η τιμή της να "περνάει" σε καταχωρητή και στην συνέχεια αφού γίνει η επεξεργασία να ξαναγυρίζει στην μεταβλητή.
- Υπάρχουν 32 καταχωρητές:
  - Ri, $0 \leq i \leq 31$,
  - **Σύμβαση**: Ο καταχωρητής R0 θα έχει πάντα την τιμή 0.

Επεξεργασία (II)

<u>Υπολογισμοί</u>:

Ri = var; /* 0 ≤ i ≤ 31, var = μεταβλητή της C */

var = Ri; /* 0 ≤ i ≤ 31, var = μεταβλητή της C */

Ri = Rj op Rk; /* 0 ≤ i,j,k ≤ 31, op = τελεστής της C */

Ri = Rj op const; /* 0 ≤ i,j,k ≤ 31, op = τελεστής της C, const = σταθερή*/

- Παραδείγματα:

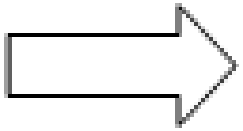
C	CLANG

int x = 5;
x = x + 6;
	int x = 5;
	R1 = x;
	R1 = R1 + 6;
	x = R1;

int x = 3, y = 7;
int z;
z = y - x;
	int x = 3, y = 7;
	int z;
	R1 = x;
	R2 = y;
	R3 = R2 – R1;
	z = R3;

Επεξεργασία Μεταβλητών

```
int x, y, z;
...
x = y * z + 3;
y = -1;
```



```
int x, y, z;
...
R3 = y;
R4 = z;
R8 = R3 * R4;
R8 = R8 + 3;
x = R8;
R1 = -1;
y = R1;
```

Έλεγχος Ροής (if)

- Χρήση Labels στην επόμενη εντολή από το if ή στην θέση του else
  - Τα Labels “ κρύβουν ” διευθύνσεις μνήμης όπου είναι αποθηκευμένη μία εντολή του προγράμματος
- Χρήση if με την **αντίστροφη συνθήκη** που περιγράφεται στο πρόγραμμα.
- Χρήση συνθήκης όπως στην C (<, <=, …)
- Χρήση **goto** για μετάβαση στην εντολή που έχει τοποθετηθεί το label.

C	CLANG

int y = 5, x = 3, z;
If(x > 3)
{
z = y + x;
}
z = z -1;
	int y = 5, x = 3, z;
	R1 = y;
	R2 = x;
	R3 = 3;
	If(R2 <= R3) goto after_if_label;
	R4 = R1 + R2;
	after_if_label:
	R4 = R4 – 1;
	z = R4;

Έλεγχος Ροής (if-else)

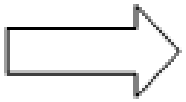
- **Παραδείγματα**

C	CLANG

int y = 5, x = 3, z;
If(x > 3)
{
z = y + x;
}
Else
{
z = y - x;
}
z = z -1;
	int y = 5, x = 3, z;
	R1 = y;
	R2 = x;
	R3 = 3;
	If(R2 <= R3) goto else_label;
	R4 = R1 + R2;
	goto after_cond;
	
	else_label:
	R4 = R1 – R2;
	
	After_cond:
	R4 = R4 – 1;
	z = R4;

CLANG: Chania C-Level Assembly Language (η «Intermediate Code»)

```
if (x < y) {
        s1;
} else {
        s2;
}
```



Αντιστροφή της συνθήκης!!!

```
        R3 = x;
        R4 = y;
        if !(R3 < R4) goto lab_else
        /* κώδικας CLANG για το s1 */
        goto lab_endif;
lab_else:
        /* κώδικας CLANG για το s2 */
lab_endif:
```

If – Else if – Else?

C	CLANG

int y = 5, x = 3, z;
If(x > 3)
{
z = y + x;
}
Else if (x == 3)
{
z = y - x;
}
Else
{
z = 5;
}
z = z -1;
	int y = 5, x = 3, z;
	R1 = y;
	R2 = x;
	R3 = 3;
	
	??

If – Else if – Else: Μετατροπή

C	CLANG

int y = 5, x = 3, z;
If(x > 3)
{
z = y + x;
}
Else
{
if (x == 3)
{
z = y - x;
}
Else
{
z = 5;
}
}
z = z -1;
	int y = 5, x = 3, z;
	R1 = y;
	R2 = x;
	R3 = 3;
	
	If(R2 <= R3) goto else_label;
	R4 = R1 + R2;
	goto after_cond;
	
	else_label:
	.....
	
	After_cond:
	R4 = R4 – 1;
	z = R4;

If – Else if – Else: Μετατροπή

C	CLANG

int y = 5, x = 3, z;
If(x > 3)
{
z = y + x;
}
Else
{
if (x == 3)
{
z = y - x;
}
Else
{
z = 5;
}
}
z = z -1;
	int y = 5, x = 3, z;
	R1 = y;
	R2 = x;
	R3 = 3;
	
	If(R2 <= R3) goto else_label;
	R4 = R1 + R2;
	goto after_cond;
	
	else_label:
	If (R2 != R3) goto else_label_2;
	R4 = R1 – R2;
	goto after_cond;
	
	else_label_2:
	R4 = 5;
	
	after_cond:
	R4 = R4 – 1;
	z = R4;

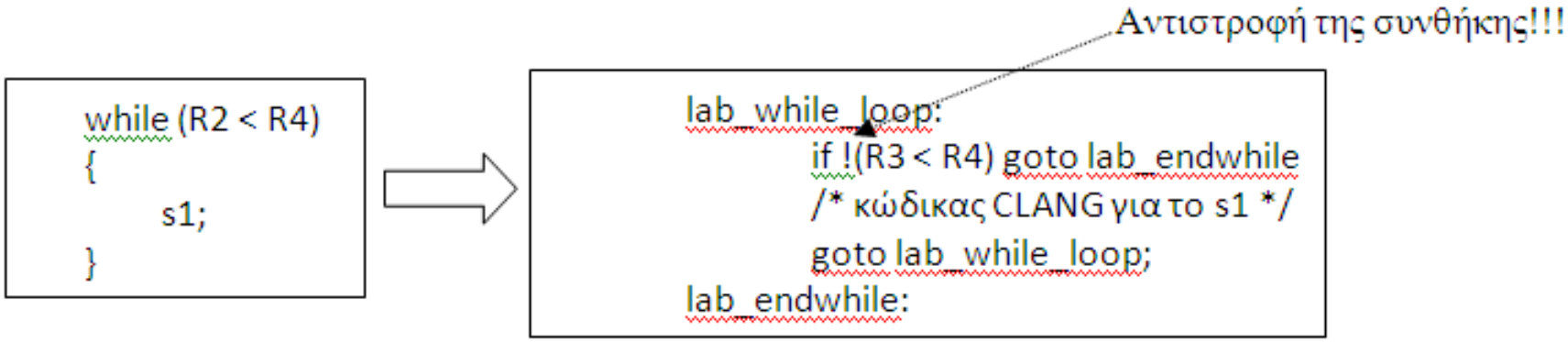
Loops (I)

- Αντικατάσταση loops με χρήση if και εντολές goto
- **While loop**:
  - while_label
  - Αντιστροφή συνθήκης
  - Goto while_label
- **Παράδειγμα**

C (while)	CLANG (while)

int y = 0, z = 20;
while(y < 3)
{
z = z - y;
y++;
}
z = z -1;
	int y = 0, z = 20;
	R1 = y;
	R2 = z;
	R3 = 3;
	While_label:
	If(R1 >= R3) goto after_loop;
	R2 = R2 – R1;
	R1 = R1 + 1;
	Goto while_label;
	after_loop:
	R2 = R2 – 1;
	y = R1; z = R2;

CLANG: Chania C-Level Assembly Language (η «Intermediate Code»)



Loops (II)

- **for loop**:
  - Μετατροπή σε while
  - Μετατροπή του while σε CLANG
- **Παράδειγμα**

C (for)

```
int  y = 0, z = 20;
for(i = 0; i < z; i++)
{
    y = y + z;
}
```

C (while)

```
int  y = 0, z = 20, i;
i = 0;
While(i < z)
{
    y = y + z;
    i++;
}
….
```

CLANG (while)

```
int y = 0, z = 20, i = 0;
R2 = z;
R3 = y;
R1 = i;
R1 = 0;
Label_loop:
If (R1 >= R2) goto after_loop;

    R3 = R3 + R2;
    R1 = R1 + 1;
    Goto Label_loop;

after_loop:
….
```

Loops (III)

- **<u>Do while loop</u>**:
  - Στην ουσία είναι while loop με λίγη διαφορετική δομή
- **Παράδειγμα**

C (do while)	CLANG (do while)
int y = 0, z = 20; **do{** **z = z - y;** **y++;** **} while(y < 3)**	int y = 0, z = 20; R1 = y; R2 = z; R3 = 3; **do_label:** **R2 = R2 – R1;** **R1 = R1 + 1;** **If(R1 >= R3) goto after_loop;** **goto do_label;** **after_loop:**

Λογικές εκφράσεις

Η αποτίμηση λογικών εκφράσεων (π.χ.)σε έλεγχο ροής) γίνεται με short-circuit evaluation:

```
if ((a < b) && (c > d)) then {
…
}

If (a < b) {
        if (c > d) {
        …
        }
}
```

Κλήση συνάρτησης

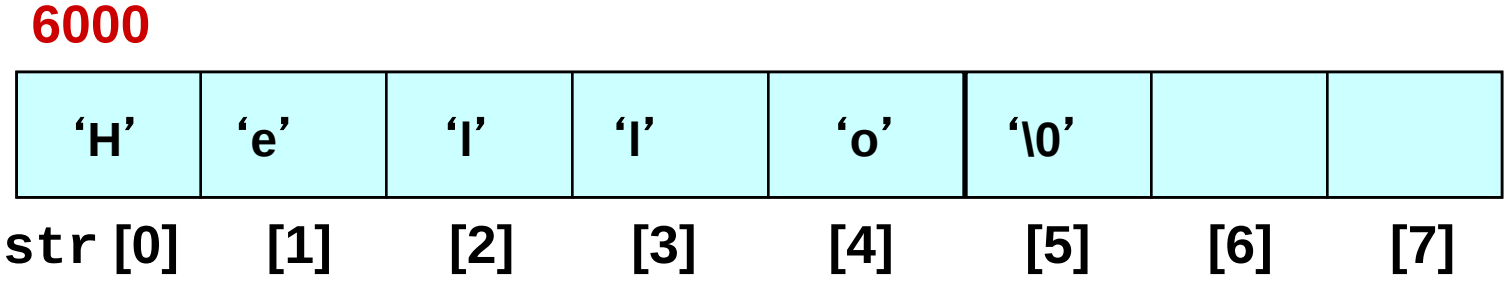
- Κλήση συνάρτησης με τον ίδιο τρόπο όπως στη C
  - Παράμετροι μόνο με τη χρήση καταχωρητών
  - Επιστρεφόμενη τιμή στον καταχωρητή R2 **($v0, σύμβαση)**
- **Παράδειγμα**

C (while)	CLANG

void main()
{
int a = 3, b = 4;
int y = foo(a, b);
}
int foo(int x, int y)
{
int z;
z = x + y;
return(z);
}
	void main() {
	int a = 3, b = 4;
	R1 = a;
	R3 = b;
	R2 = foo(R1, R3);
	y = R2;
	}
	int foo(int R1, int R3)
	{
	R2 = R1 + R3;
	return(R2);
	}

Θέματα Μνήμης

- C: Array 1D➔ CLANG: Global Array 1D
- C: Array 2D➔ CLANG: Global Array 1D (Μετατροπή του 2D όπως σας παρουσιάστηκε στο προηγούμενο μάθημα)
- **Παραδείγματα**
  - char str [8];



- Example 3 x 4 array:

a b c d

e f g h

i j k l

- Convert into 1D array y by collecting elements by columns.
- We get y = {a, e, i, b, f, j, c, g, k, d, h, l}

- Example 3 x 4 array:

a b c d

e f g h

i j k l

- Convert into 1D array y by collecting elements by rows.
- We get y[] = {a, b, c, d, e, f, g, h, i, j, k, l}

Χρήση Πινάκων (Ι)

- Στη CLANG οι πίνακες πρέπει να ορίζονται ως global μεταβλητές.
- Όλοι οι πίνακες στην CLANG καλό είναι να χρησιμοποιείτε 1D arrays.
- Η πρόσβαση στους πίνακες της CLANG θα γίνεται με χρήση αριθμητικής διευθύσεων.
- Παραδείγματα:

C	CLANG
int x[3] = {1, 2, 3}; x[2] = x[1] + 3;	int x[3] = {1, 2, 3}; R1 = 1; R2 = x[R1]; R3 = R2 + 3; R2 =2; x[R2] = R3;

Χρήση Πινάκων (II)

C	CLANG
int x[2][3] = {{1, 2, 3}, {4,5,6}}; x[1][2] = x[1][1] + 5;	int x[6] = {1, 2, 3, 4, 5, 6}; R1 = 1; R2 = 2; R3 = R1*3 + R1; R4 = x[R3]; R5 = R4 + 5; R3 = R1*3 + R2; X[R3] = R5;

I/O functions

- Οι συναρτήσεις που θα χρησιμοποιείτε θα είναι ίδιες με αυτές της C.

C	CLANG
int x = 5; printf(“%d\n”, x);	int x = 5; R1 = x; printf(“%d\n”, R1);
int x; scanf(“%d\n”, &x)	int x; scanf(“%d\n”, &R1) x = R1;

C Παράδειγμα 1: Add two Arrays

```
int a[4] = {1, 2, 3, 4};
int b[4] = {15, 16, 17, 18};
int sum[4];

int main()
{
	int i = 0;

	for (i = 0; i < 4; i++)
	{
		sum[i] = a[i] + b[i];
	}

	for (i = 0; i < 4; i++)
	{
		printf("%d\n",sum[i]);
	}
	return(0);
}
```

CLANG Παράδειγμα 1: Add two Arrays (I)

```
int a[4] = {1, 2, 3, 4};
int b[4] = {15, 16, 17, 18};
int sum[4];

int R0 = 0;

int R1, R2, R3, R4, R5, R6, R7, R8, R9, R10, R11, R12, R13, R14, R15, R16, R17, R18, R19, R20, R21, R22, R23, R24, R25,
      R26, R27, R28, R29, R30, R31;

void main()
{
      int i = 0;

      R1 = i;

      R1 = 0;
      R2 = 4;

while_label: if(R1 >= R2) goto end_loop;

      R3 = a[R1];
      R4 = b[R1];
      R5 = R3 + R4;
      sum[R1] = R5;

      R1 = R1 + 1;
      goto while_label;
```

CLANG Παράδειγμα 1: Add two Arrays (II)

```
end_loop:

        R1 = 0;
while_label2: if(R1 >= R2) goto end_loop2;

        R3 = sum[R1];
        printf("%d\n",R3);

        R1 = R1 + 1;
        goto while_label2;

end_loop2:
        i = 4;

}
```

C Παράδειγμα 2: Add two Matrices

```
int a[3][4] = {{1,2,3,4},{4,5,6,7},{7,8,9,10}};
int b[3][4] = {{33,34,35,35},{36,37,38,39},{39,40,41,42}};
int result[3][4];

void main()
{
	int i = 0, j =0;

	for(i = 0; i < 3; i++)
	{
		for(j = 0; j < 4; j++)
		{
			result[i][j] =  a[i][j] + b[i][j];
		}
	}

	for(i = 0; i < 3; i++)
	{
		for(j = 0; j < 4; j++)
		{
			printf("%d\n", result[i][j]);
		}
	}
}
```

CLANG Παράδειγμα 2: Add two Matrices

```
#include <stdio.h>

int a[12] = {1,2,3,4,4,5,6,7,7,8,9,10};
int b[12] = {33,34,35,35,36,37,38,39,39,40,41,42};
int result[12];


int R0 = 0;

int R1, R2, R3, R4, R5, R6, R7, R8, R9, R10, R11, R12, R13, R14, R15, R16, R17, R18, R19, R20, R21, R22, R23, R24, R25,
    R26, R27, R28, R29, R30, R31;
void main()
{
    int i = 0, j =0;

    R1 = i;
    R2 = j;
    R3 = 3;
    R4 = 4;

   for_loop1:
        if ( R1 >= R3) goto end_loop1;

        R2 = 0;

        for_loop2: if( R2 >= R4) goto end_loop2;
```

CLANG Παράδειγμα 2: Add two Matrices

```
                R5 = R1*R4 + R2;
                R6 = a[R5];
                R7 = b[R5];

                R8 = R6 + R7;

                result[R5] = R8;

                R2 = R2 + 1;
                goto for_loop2;

        end_loop2:

        R1 = R1 + 1;
        goto for_loop1;

end_loop1:
```

CLANG Παράδειγμα 2: Add two Matrices

```
R1 = 0;
    for_loop3:
        if ( R1 >= R3) goto end_loop3;

        R2 = 0;

        for_loop4: if( R2 >= R4) goto end_loop4;

                R5 = R1*R4 + R2;
                R6 = result[R5];
                printf("%d\n",R6);

                R2 = R2 + 1;
                goto for_loop4;

        end_loop4:

        R1 = R1 + 1;
        goto for_loop3;

    end_loop3:
}
```

C Παράδειγμα 3: Removes double spaces or blanks from a string

```
#include <stdio.h>
int main()
{
	char text[100], blank[100];
	int c = 0, d = 0;
	printf("Enter some text\n");
	gets(text);
	while (text[c] != '\0')
	{
		if (text[c] != ' ' || text[c+1] != ' ')
		{
			blank[d] = text[c];
			d++;
		}
		c++;
	}
	blank[d] = '\0';
	printf("Text after removing blanks\n%s\n", blank);
	return 0;
}
```

CLANG Παράδειγμα 3: Removes double spaces or blanks from a string

```
#include <stdio.h>

int R0=0, R1, R2, R3, R4, R5, R6, R7, R8, R9, R10,
   R11, R12, R13, R14, R15, R16, R17, R18, R19, R20,
   R21, R22, R23, R24, R25, R26, R27, R28, R29, R30,
   R31;

char text[100];
char blank[100];


int main()
{
      int c = 0, d = 0;

      R1 = c;
      R2 = d;


      printf("Enter some text\n");
      gets(text);
```

CLANG Παράδειγμα 3: Removes double spaces or blanks from a string

```
label_while:
                        R3 = text[R1];
                        if(R3 == '\0') goto end_while;

                                R4 = text[R1];
                                R5 = R1 + 1;
                                R6 = text[R5];

                                if (R4 == ' ') goto second_codition;
                                        goto if_code;
                                        second_condition: if(R6 == ' ') goto end_if;
                                        if_code:
                                                blank[R2] = R4;
                                                R2 = R2 + 1;
                                end_if:
                        R1 = R1 + 1;
                        goto label_while;

        blank[R2] = '\0';
        printf("Text after removing blanks\n%s\n", blank);
        return 0;
}
```

C Παράδειγμα 4: Condition **AND**

```
int x = 5, y = 8, z;

if(x <= 7 && y > 10)
    z = x + y;
else
    z = x – y;
```

CLANG Παράδειγμα 4: Condition **AND**

```
int x = 5, y = 8, z;
R1 = x;
R2 = y;
    if ( R1 > 7) goto else_label;
    If (R2 <= 10) goto else_label;
            R3 = R1 + R2;
            goto end_if;
   else_label:
            R3 = R1 – R2;
End_if:
            z = R3;
```

C Παράδειγμα 5: Αναδρομή

```
#include<stdio.h>


int Fibonacci(int n)
{
  if ( n == 0 )
    return 0;
  else if ( n == 1 )
    return 1;
  else
    return ( Fibonacci(n-1) + Fibonacci(n-2) );
}
```

CLANG Παράδειγμα 5: Αναδρομή

```
int Fibonacci(int R2)
{
   int temp1;

   if(R2 != 0) goto lab_else_if;
      R9 = 0;
      goto lab_end;
   lab_else_if:
   if (R2 != 1 ) goto lab_else;
      R9 = 1;
      goto lab_end;
   lab_else:

      R5 = R2-1;
      R7 = Fibonacci(R5);
      temp1 = R7;

      R6 = R2-2;
      R8 = Fibonacci(R6);
      R7 = temp1;
      R9 = R7+R8;

   lab_end:
      return(R9);
}
```

Definitions

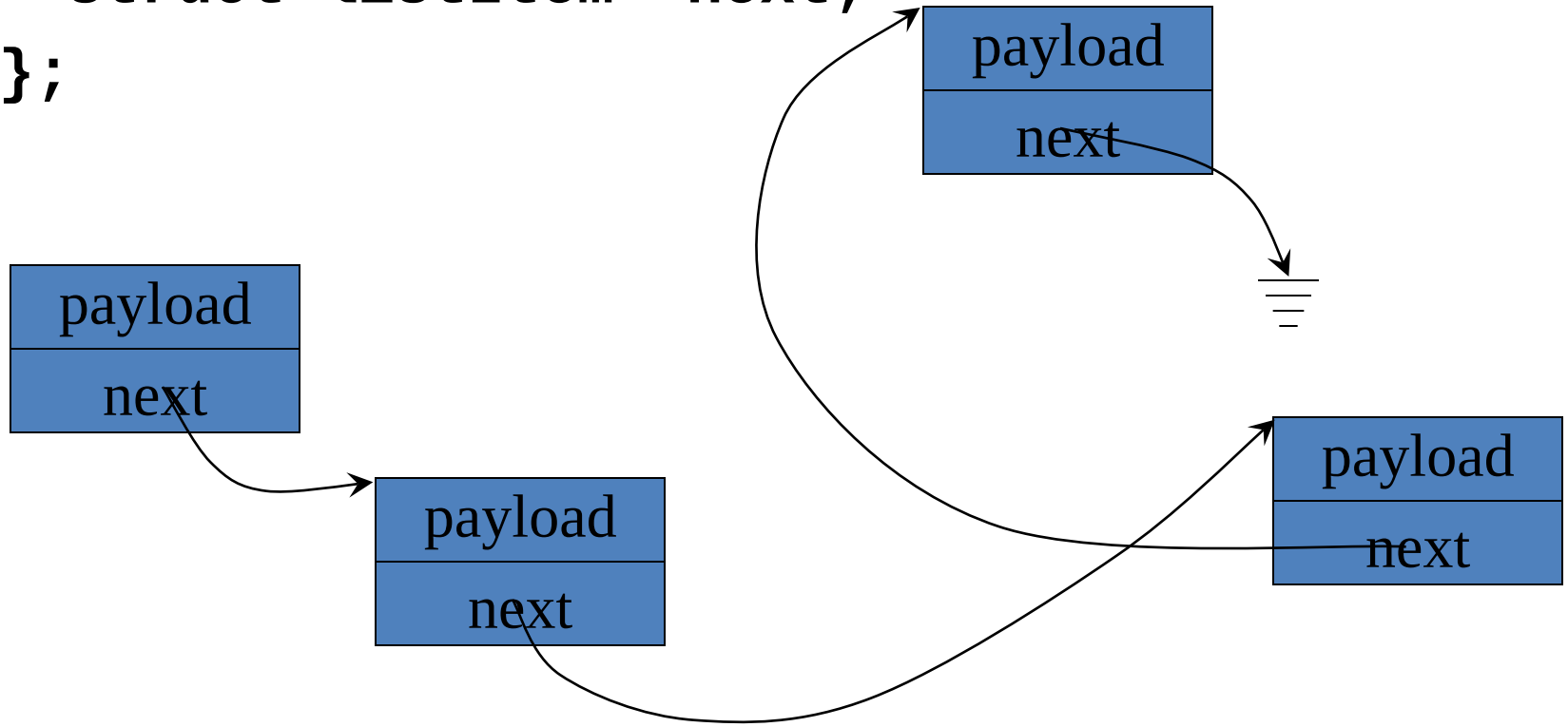
Note: elements are *usually* the same type (but not always).

- *Linked List*
  - A *data structure* in which each element is dynamically allocated and in which elements point to each other to define a *linear* relationship
  - Singly- or doubly-linked
  - Stack, queue, circular list
- *Tree*
  - A *data structure* in which each element is dynamically allocated and in which each element has more than one potential *successor*
  - Defines a *partial order*

Linked List

```
struct listItem {
  type payload;
  struct listItem *next;
};
```

Note: payload may be multiple members.



Linked List (continued)

- Items of list are usually same type
  - Generally obtained from **`malloc()`**
- Each item points to next item
- Last item points to null
- Need "**`head`**" to point to first item!

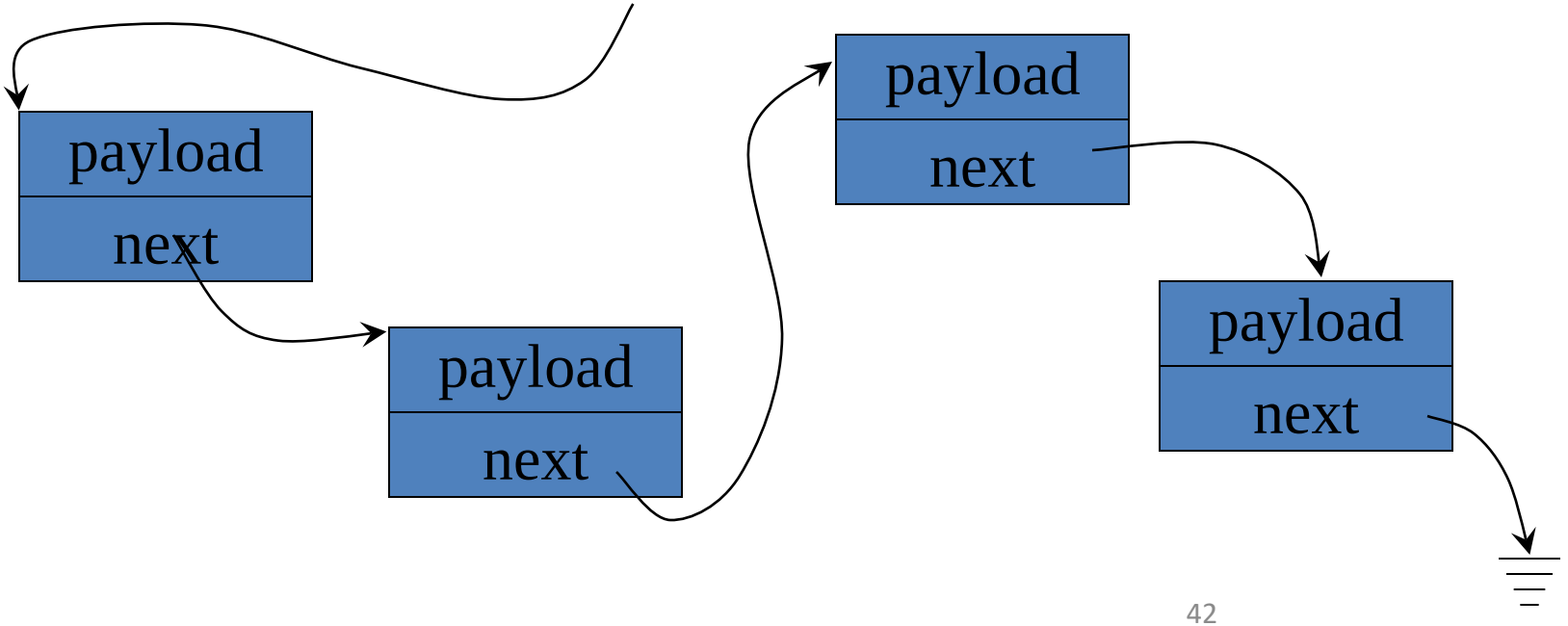
- "Payload" of item may be almost anything
  - A single member or multiple members
  - Any type of object whose size is known at compile time
  - Including **`struct`**, **`union`**, **`char *`** or other pointers
  - Also arrays of fixed size at compile time

Usage of Linked Lists

- Not massive amounts of data
  - Linear search is okay
- Sorting not necessary
  - or sometimes not possible
- Need to add and delete data “on the fly”
  - Even from middle of list
- Items often need to be added to or deleted from the “ends”

Linked List (continued)

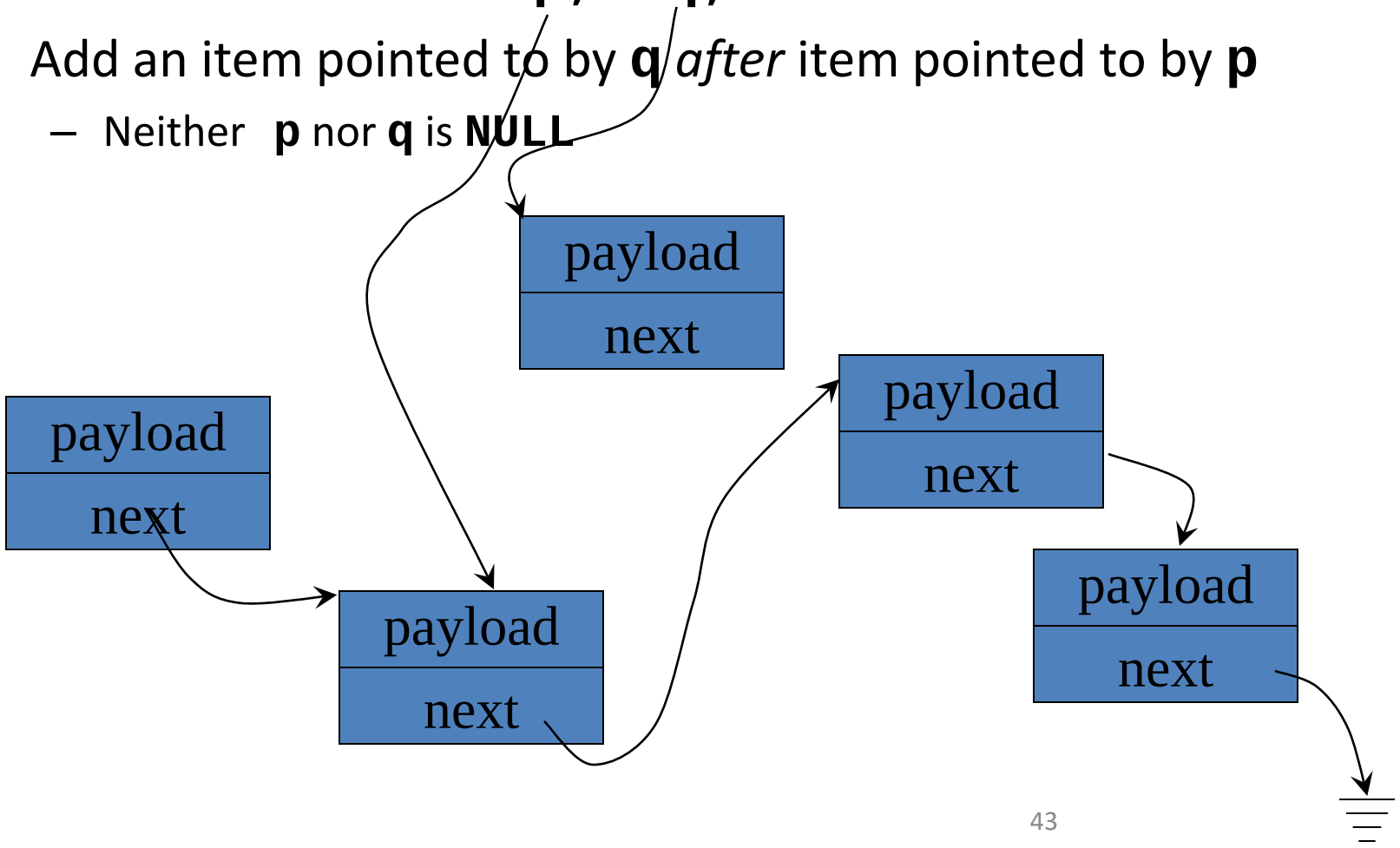
```
struct listItem {
  type payload;
  struct listItem *next;
};
struct listItem *head;
```



Adding an Item to a List

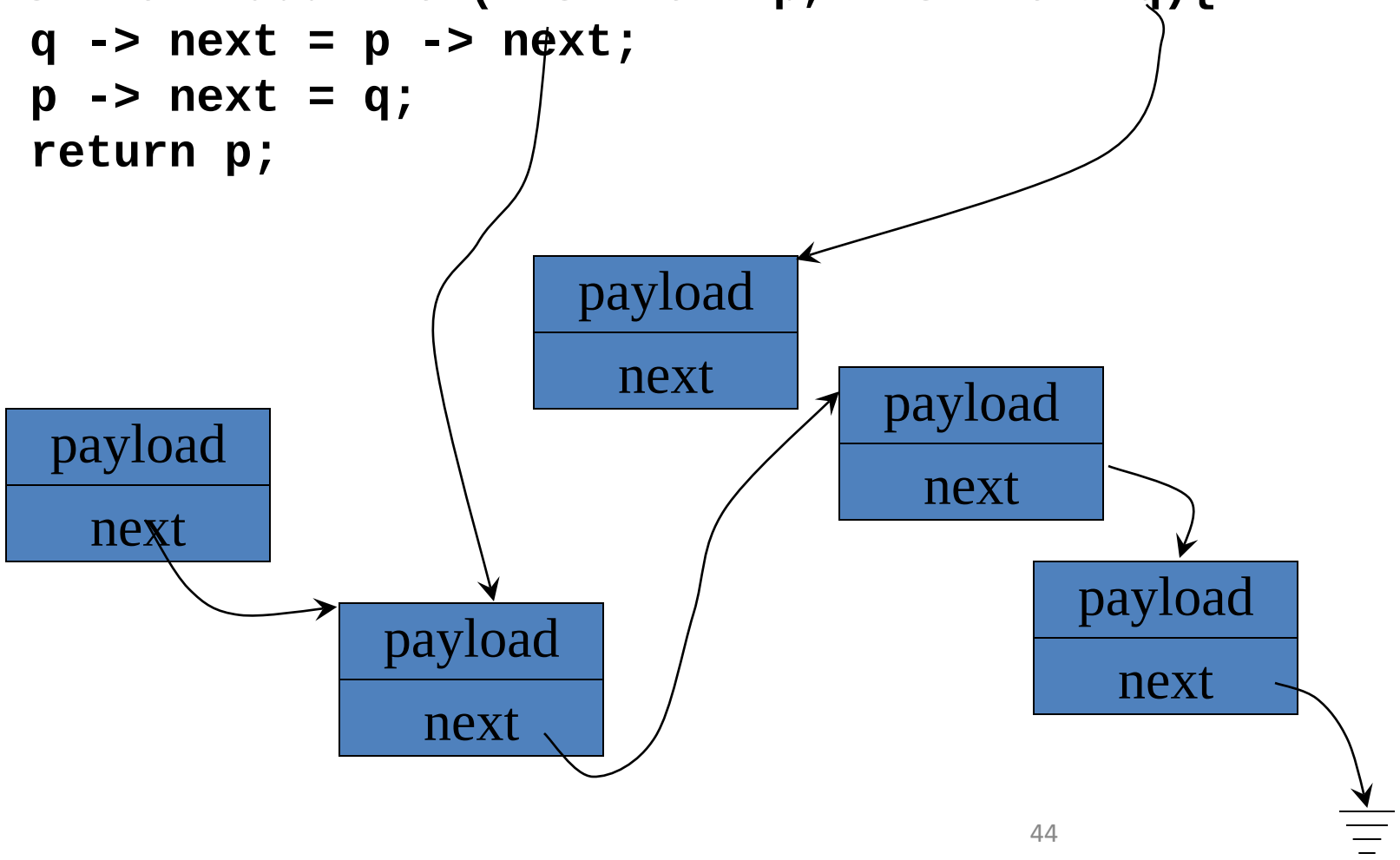
```
struct listItem *p, *q;
```

- Add an item pointed to by **q** *after* item pointed to by **p**
  - Neither **p** nor **q** is **NULL**



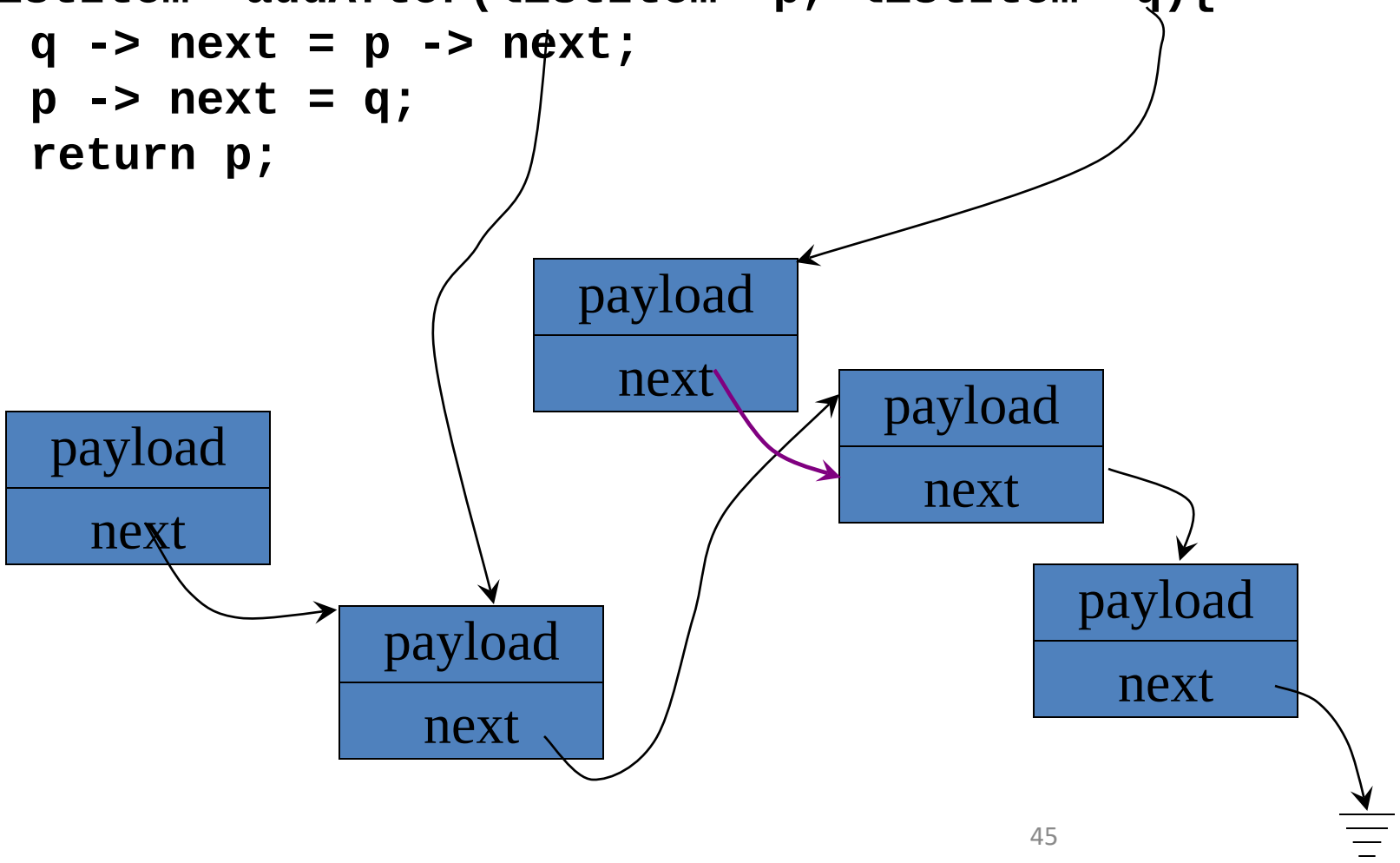
Adding an Item to a List

```
listItem *addAfter(listItem *p, listItem *q){
   q -> next = p -> next;
   p -> next = q;
   return p;
}
```



Adding an Item to a List

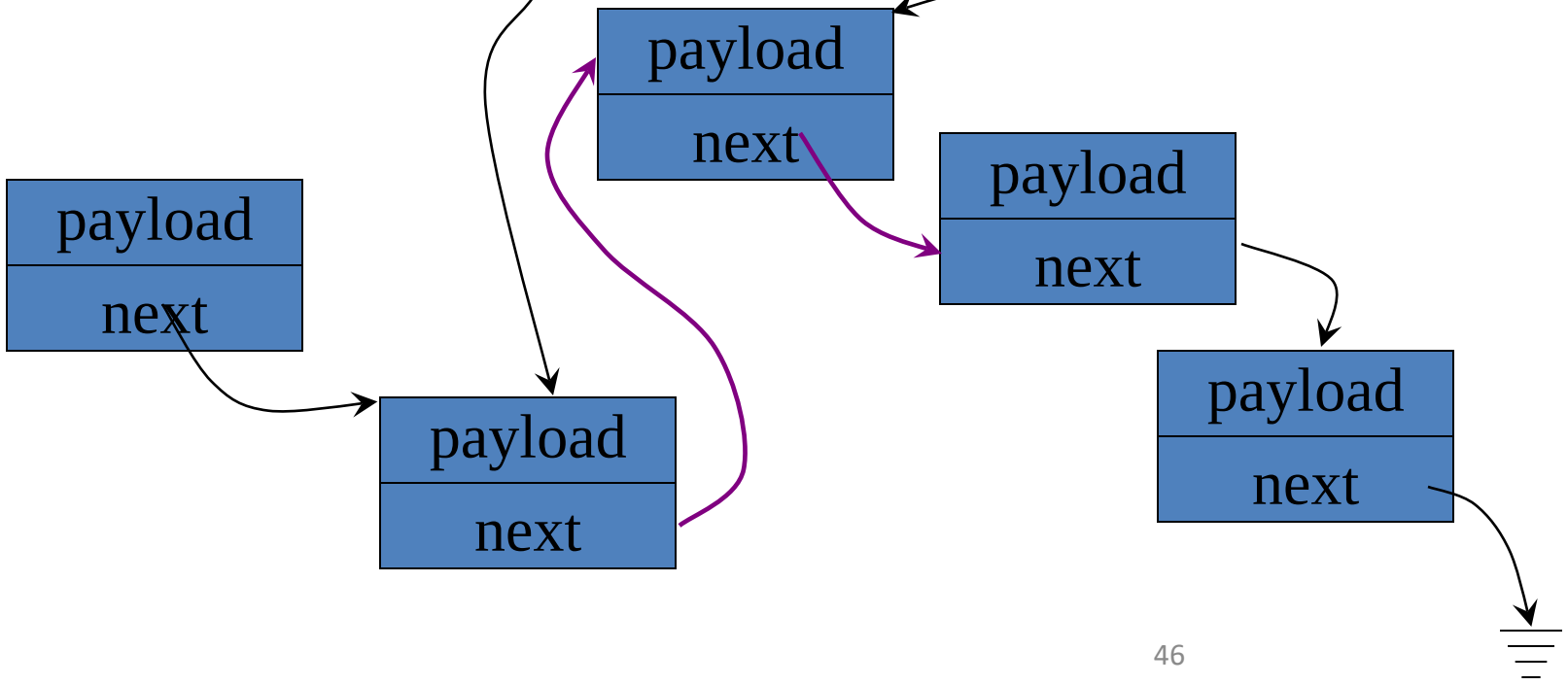
```
listItem *addAfter(listItem *p, listItem *q){
   q -> next = p -> next;
   p -> next = q;
   return p;
}
```



Adding an Item to a List

Question: What to do if we cannot guarantee that *p* and *q* are non-NULL?

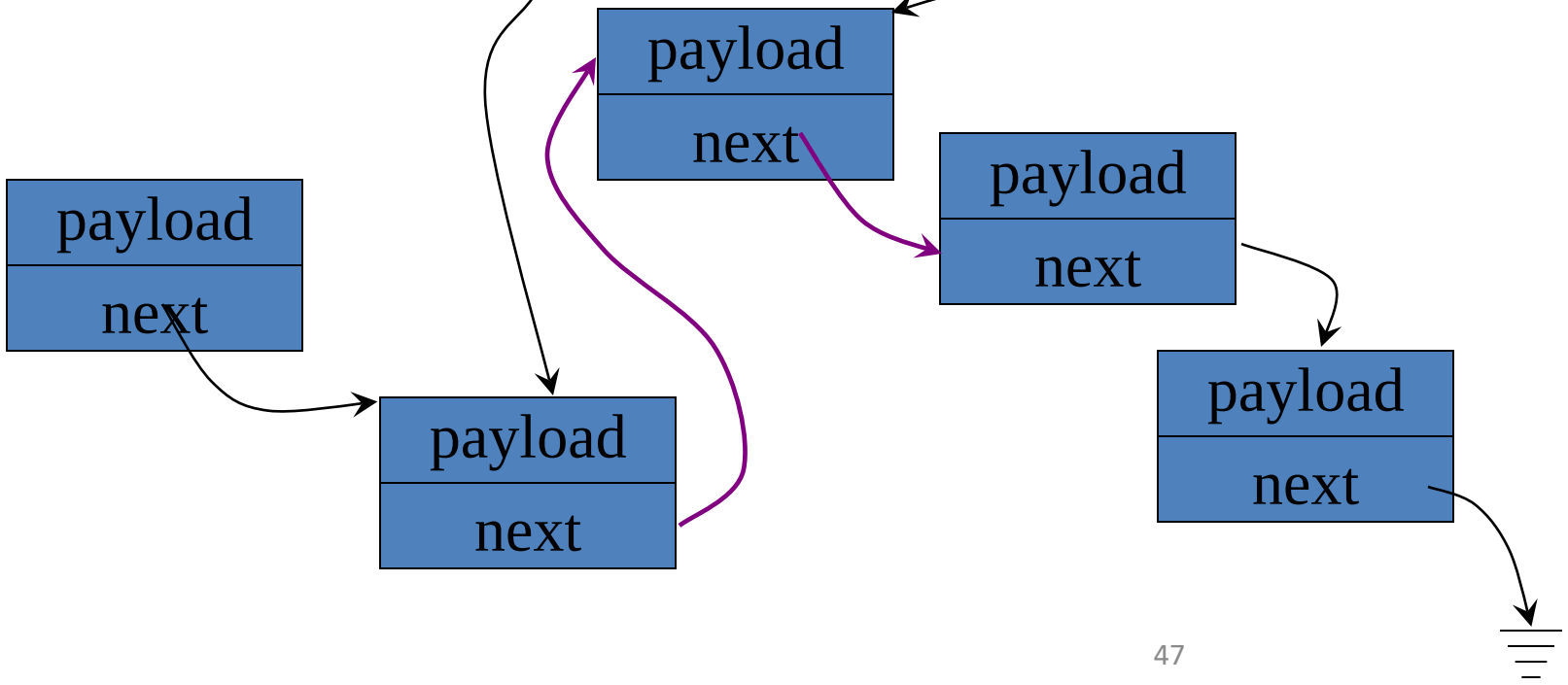
```
listItem *addAfter(listItem *p, listItem *q){
   q -> next = p -> next;
   p -> next = q;
   return p;
}
```



Adding an Item to a List (continued)

Note test for non-null p and q

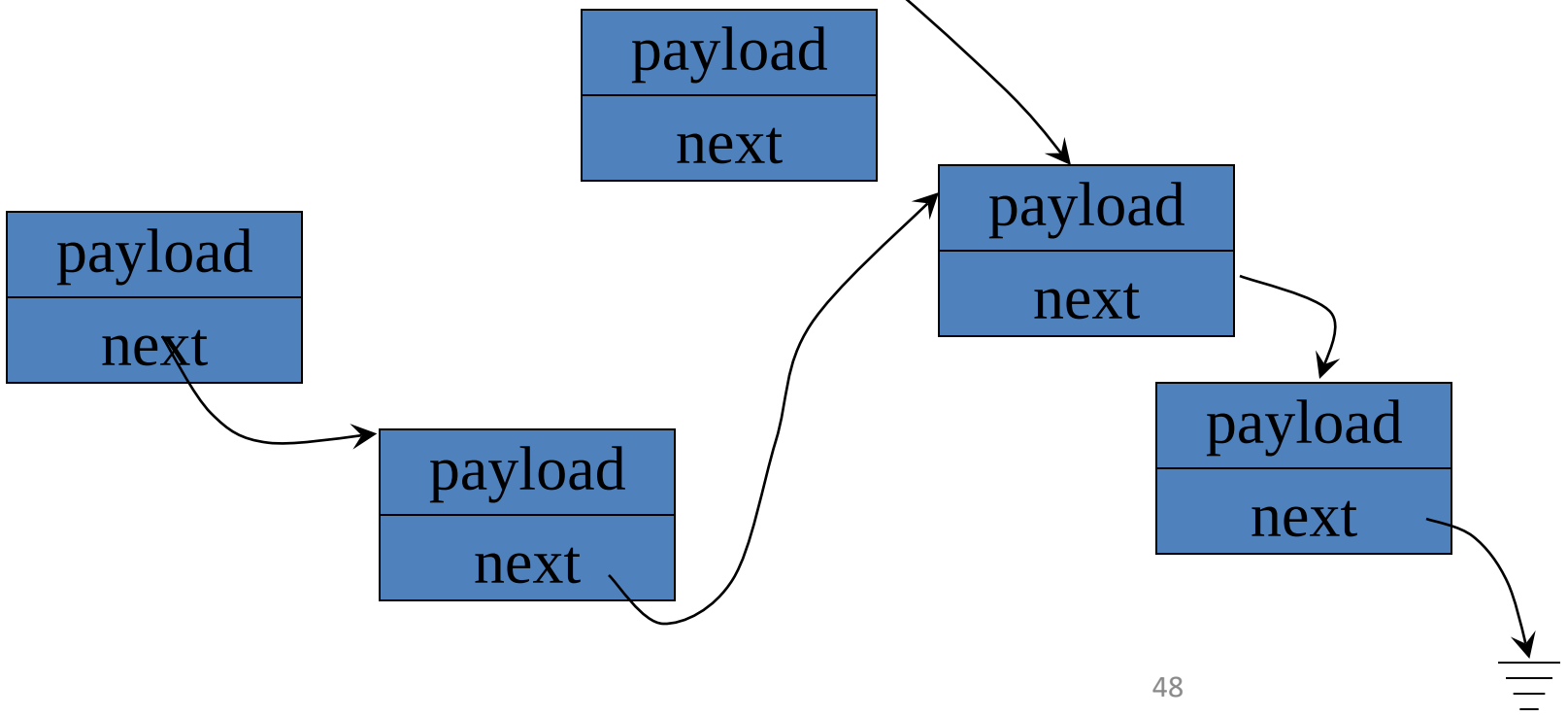
```
listItem *addAfter(listItem *p, listItem *q){
   if (p && q) {
       q -> next = p -> next;
       p -> next = q;
   }
   return p;
}
```



What about Adding an Item *before* another Item?

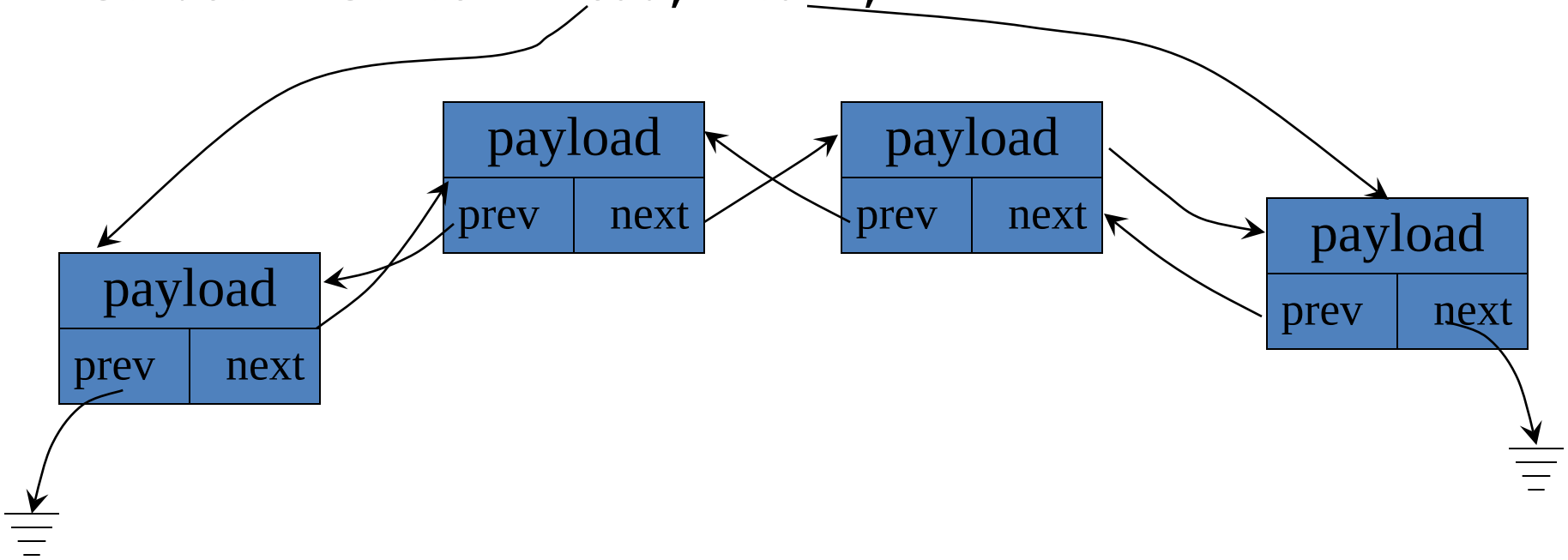
```
struct listItem *p;
```

- Add an item *before* item pointed to by **`p`** (**`p != NULL`**)



Doubly-Linked List

```
struct listItem {
   type payload;
   listItem *prev;
   listItem *next;
};
struct listItem *head, *tail;
```



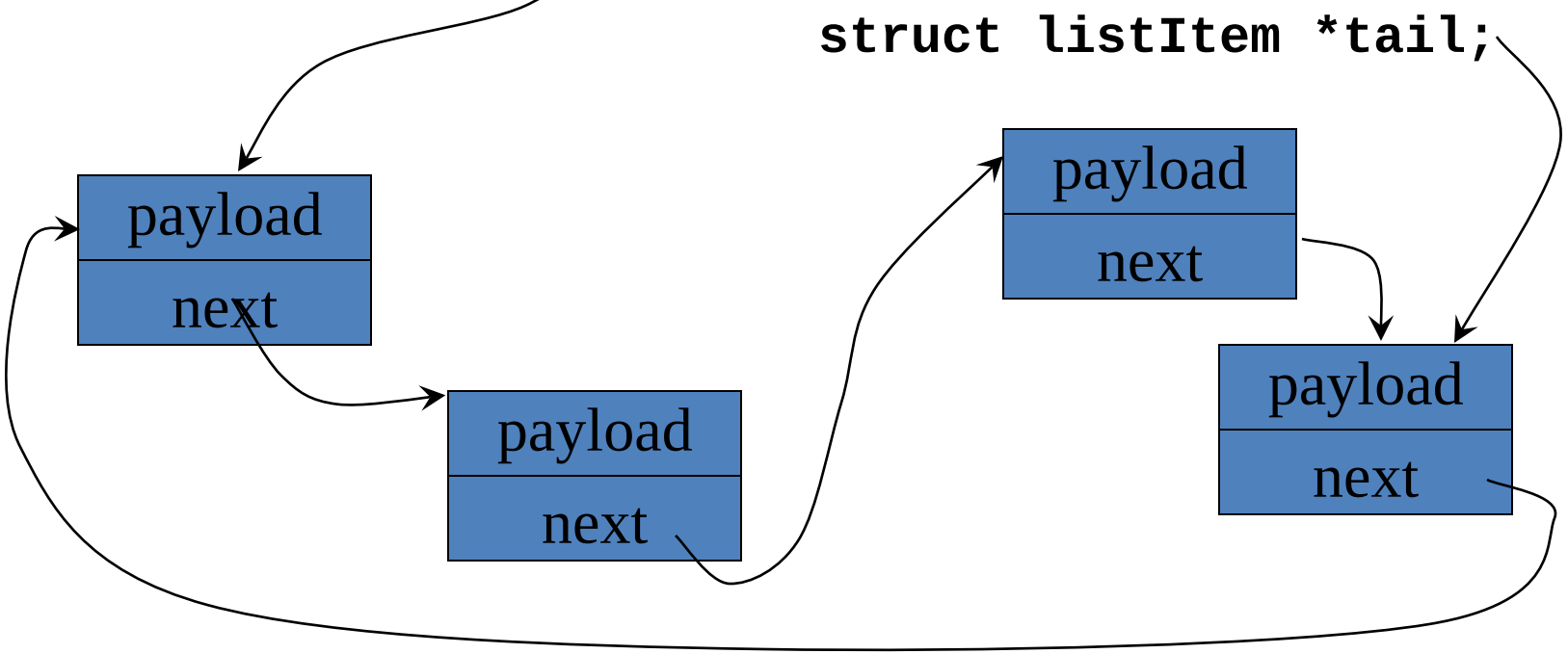
Other Kinds of List Structures

- *Queue* — FIFO (First In, First Out)
  - Items added at *end*
  - Items removed from *beginning*
- *Stack* — LIFO (Last In, First Out)
  - Items added at *beginning*, removed from *beginning*
- *Circular list*
  - Last item points to first item
  - Head may point to first or last item
  - Items added to *end*, removed from *beginning*

Optional:–
`struct listItem *head;`

Circular List

`struct listItem *tail;`




```
listItem *addAfter (listItem *p, listItem *tail){
   if (p && tail) {
        p -> next = tail -> next;
        tail = p;
   } else if (p) {
        tail p -> next = p;
   }
   return tail;
}
```