

C struct, pointer, and linked list



Umi Sa'adah

PENS – 2016

Overview

- Structure
 - Nested structure, pointer to structure, structure as parameter
- Pointer
 - Dynamic variables, malloc, free, NULL
- Linked list
 - Add / Delete a node, traverse a linked list
- Modules

Structure

```
typedef struct {  
    int day, month, year;  
} Date;
```

```
main ( ) {  
    Date birthday  
        = { 18, 2, 1999 };  
    Date today;  
  
    today.day = 28;  
    today.month = 2;  
    today.year = 2002;  
    ...  
}
```

	today
today.day	28
today.month	2
today.year	2002

	birthday
birthday.day	18
birthday.month	2
birthday.year	1999

Structure

- Mengelompokkan field-field yang berelasi menjadi sebuah struct

```
struct date {  
    int day, month, year;  
};  
struct date today;
```

```
typedef struct {  
    int day, month, year;  
} Date;
```

Date today;

```
typedef struct date {  
    int day, month, year;  
} Date;
```

struct date today;
or
Date today;

Mengakses field dalam struct

```
Date today;  
today.day = 28;  
today.month = 2;  
today.year = 2002;  
printf("This year is %d", today.year);
```

Untuk mengakses sebuah field,
digunakan operator titik. Contoh :
today.year adalah field year dari
struct today

Nested Struct, Array of Struct

```
typedef struct {  
    int d, m, y;  
} Date;
```

```
typedef struct {  
    char name[60];  
    double averageScore;  
    Date dob;           //date of birth  
} Student;
```

```
typedef struct class{  
    Student stud[30];  
    int numStud;  
}Class;
```


Nested Record, Array of Record

```
...  
int main ( ) {  
    Class mccs170;  
    Date someday;  
    mccs170.numStud = 12;  
    strcpy(mccs170.stud[0].name, "Peter");  
    mccs170.stud[0].averageScore = 88;  
    someday.d = 1;  
    someday.m = 1;  
    someday.y = 1980;  
    mccs170.stud[0].dob = someday;  
    ...  
}
```

Sebuah struct bisa dicopy ke struct yang lain menggunakan operator '='

Exercise

Q1) Write statements to initialize the record of student no. 1 as follows.

Name: "Tommy"

Average score: 91

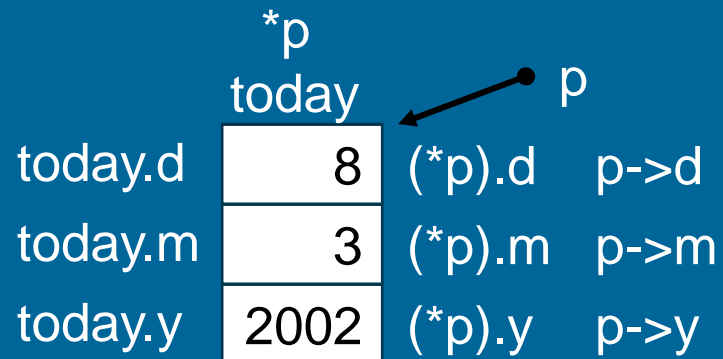
Date of birth: 2 Apr 1980

```
int main ( ) {  
    Class mcs170;  
    ...  
}
```


Pointer to Structure

```
typedef struct {
    int d, m, y;
} Date;

main () {
    Date today = { 8, 3, 2002 };
    Date *p;
    p = &today;
    ...
}
```



Perhatikan :
***p.d** artinya
***(p.d)**, bukan
(*p).d.

Structure as Input Parameter

```
typedef struct {
    int d, m, y;
} Date;

main () {
    Date today = { 8, 3, 2002 };
    print_date(today);
    ...
}

void print_date (Date aday)
{
    printf ("%d/%d/%d",
        aday.d, aday.m, aday.y);
}
```

Passing struct sebagai parameter adalah sama dengan passing variabel integer. aday dan today adalah dua variabel yang terpisah

Exercise

Q2) Tuliskan fungsi di bawah ini

```
struct Date { int d, m, y; };
```

```
void print_date2 (Date aday) {  
    char months[12][4] = {  
        "Jan", "Feb", "Mar", ...  
        "Nov", "Dec" };  
    //print the date in the format 25 Dec 2000  
}
```

Exercise

Q3) Tuliskan fungsi di bawah ini yang mempunyai return value hari dalam seminggu

```
struct Date { int d, m, y; };  
  
int dayofweek (Date aday) {  
    ...  
}
```

Exercise

Q3 cont) Gunakan algoritma berikut ini untuk mendapatkan nama hari dalam seminggu.

```
w = (d+(13*m-1)/5 - 2*c + y + y/4 + c/4) % 7  
if (w<0) w+=7;
```

where

d is the day (1-31)

m is month (1=Mar, ..., 10=Dec, 11=Jan, 12=Feb)

Treat Jan and Feb as month of the preceding year

y is year (2000 has y = 00 except y = 99 for Jan and Feb),

c is century (1999 has c = 19. 2000 has c = 20 except c = 19 for Jan and Feb),

w is week day (0 = Sunday, ..., 6 = Saturday).

Struct sebagai Output Parameter

```
main () {  
    Date today;  
    scan_date(&today);  
    print_date(today);  
    ...  
}
```

Dilakukan pengiriman pointer to struct (*pass by reference*) kepada sebuah fungsi yang akan memodifikasi struct, sebagaimana pada bilangan integer.

```
void scan_date (Date *aday) {  
    int dd, mm, yy;  
    scanf("%d %d %d", &dd, &mm, &yy);  
    (*aday).d = dd;  
    (*aday).m = mm;  
    (*aday).y = yy;  
}
```


Pointer to Structure

```
void scan_date (Date *aday) {  
    int dd, mm, yy;  
    scanf("%d %d %d", &dd, &mm, &yy);  
    aday->d = dd; —  
    aday->m = mm;  
    aday->y = yy;  
}
```

A shorthand:

(*aday).d same as aday->d
(*aday).m same as aday->m

Pointer to Structure

```
void scan_date (Date *aday) {  
    scanf( "%d %d %d" ,  
        &aday->d,  
        &aday->m,  
        &aday->y) ;  
}
```

Jika ingin mengeset nilai field dalam struct menggunakan *scanf()*, harus disertakan addressnya (operator &)

Structure as Parameter

```
void print_date (Date aday) {  
    printf("%d/%d/%d",  
        aday.d, aday.m, aday.y);  
}  
  
void scan_date (Date *aday) {  
    scanf("%d %d %d", &aday->d,  
        &aday->m, &aday->y);  
}
```

Comparing content of structure

```
typedef struct {  
    int d, m, y;  
} Date;  
  
main ( ) {  
    Date day1 = { 11, 2, 1999 };  
    Date day2 = { 1, 12, 1999 };  
    ...  
    if (day1==day2) ✗  
        ...  
}
```

Meskipun bisa menggunakan "="
utk mengcopy struct, namun tidak
bisa membandingkan struct
dengan "==".

Exercise

```
typedef struct {  
    int d, m, y;  
} Date;
```

```
typedef struct {  
    char name[60];  
    double averageScore;  
    Date dob; //date of birth  
} Student;
```

```
struct Class {  
    Student stud[30];  
    int numStud;  
};
```

Latihan-latihan
selanjutnya menggunakan
asumsi-asumsi berikut.

Exercise

```
main ( ) {  
    Date day1, day2;  
    ...  
    if (sameday(day1,day2))  
        printf("It is the same day\n");  
}  
int sameDay (Date day1, Date day2) {  
    ...  
}
```

Q4) Tuliskan sebuah fungsi untuk menentukan apakah 2 buah tanggal sama.

Exercise

Q5) Tuliskan fungsi untuk mengecek apakah student 1 lebih muda dari student 2.

```
int isYounger (Student stud1, Student stud2) {  
    ..  
}
```

Static Variable

```
int isPrime (int x) {  
    for (int k=2; k<x/2; k++)  
        if (x%k==0) return 0;  
    return 1;  
}  
int sumPrime (int N) {  
    int sum = 0;  
    for (int i=1; i<=N; i++)  
        if (isPrime(i)) sum+=i;  
    return sum;  
}  
int main ( ) {  
    int ans = sumPrime(10);  
}
```

Storage area untuk variabel local dan global dialokasikan dan dibebaskan secara otomatis. Biasa disebut variabel static. Variabel local dibuat ketika pemanggilan fungsi dan dihancurkan ketika fungsi kembali.

Static Variable

```
int isPrime (int x) {  
    for (int k=2; k<x/2; k++)  
        if (x%k==0) return 0;  
    return 1;  
}  
int sumPrime (int N) {  
    int sum = 0;  
    for (int i=1; i<=N; i++)  
        if (isPrime(i)) sum+=i;  
    return sum;  
}  
int main ( ) {  
    int ans = sumPrime(10);  
}
```

Local variables of isPrime

x	<input type="text"/>
k	<input type="text"/>

Local variables of sumPrime

N	<input type="text" value="10"/>
sum	<input type="text" value="0"/>
i	<input type="text"/>

Local variables of main

ans	<input type="text"/>
-----	----------------------

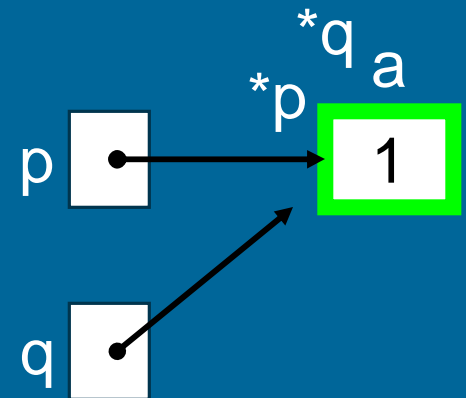
Weakness of Static Variable

```
// refer to slide 6 for detail
struct Class{
    Student stud[30];
    int numStud;
};
```

Ukuran storage dikontrol oleh C compiler. Kita harus secara eksplisit “**hardcode the number and size**” of variables dalam program. Misal, diberikan bilangan (30) dari *student*, dan gunakan *numStud* untuk menyimpan jumlah sesungguhnya dari students. Jika jumlahnya ternyata lebih sedikit, akan terjadi memori yang sia-sia. Jika ternyata lebih dari 30, tidak semua data bisa disimpan.

Pointer and Static Variable

```
#include <stdio.h>
void foo () {
    int *p, *q; int a=1;
    q = &a; p = q;
    *q = 3; *p = 5;
    printf("%d %d %d", a, *p, *q);
}
int main ( ) { ... foo(); ... }
```



A pointer to a static variable hanya sekedar memberikan alias kepada variabel tsb. Hal ini tidak berpengaruh terhadap allocation and de-allocation dari storage space.

Pointer and Dynamic Variable

The true power of pointer is in the usage of dynamic variables and different data structures.

Need of Dynamic Variable

```
void student_register ( ... ) {  
    // allocate space for the new student  
}  
  
void student_graduate (char name[ ]) {  
    // free the space for the graduated student  
}
```

Terkadang tidak diketahui berapa banyak kebutuhan memori ketika menulis program. We may want to allocate space in a function and free the space in another function. We need a more flexible way to manage storage space.

Dynamic Variables

Dynamic variables dibuat dan dihancurkan secara manual. We call

- `malloc()` to allocate, and
- `free()` to free

storage space of a dynamic variable.

Dynamic variable hanya bisa diakses melalui pointer.

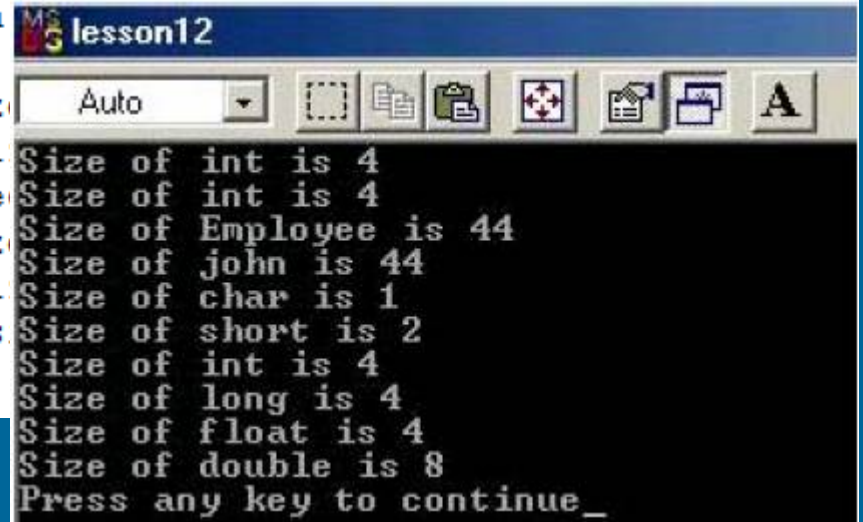
malloc

- Menyediakan fasilitas untuk membuat ukuran buffer dan array secara dinamik.
- Dinamik artinya bahwa ruang dalam memori akan dialokasikan ketika program dieksekusi (run time)
- Fasilitas ini memungkinkan user untuk membuat tipe data dan struktur dengan ukuran dan panjang berapapun yang disesuaikan dengan kebutuhan di dalam program.

sizeof()

- Digunakan untuk mendapatkan ukuran dari berbagai tipe data, variabel ataupun struktur.
 - Return value : ukuran dari obyek yang bersangkutan dalam byte.
 - Parameter dari sizeof() : sebuah obyek atau sebuah tipe data

```
typedef struct employee_st {  
    char name[40];  
    int id;  
} Employee;  
  
main() {  
    int myInt;  
    Employee john;  
  
    printf("Size of int is %d\n", sizeof(myInt));  
    //The argument of sizeof is an object  
  
    printf("Size of int is %d\n", sizeof(int));  
    //The argument of sizeof is a data type  
  
    printf("Size of Employee is %d\n", sizeof(Employee));  
    //The argument of sizeof is an object  
  
    printf("Size of john is %d\n", sizeof(john));  
    //The argument of sizeof is a  
  
    printf("Size of char is %d\n", sizeof(char));  
    printf("Size of short is %d\n", sizeof(short));  
    printf("Size of int is %d\n", sizeof(int));  
    printf("Size of long is %d\n", sizeof(long));  
    printf("Size of float is %d\n", sizeof(float));  
    printf("Size of double is %d\n", sizeof(double));  
}
```

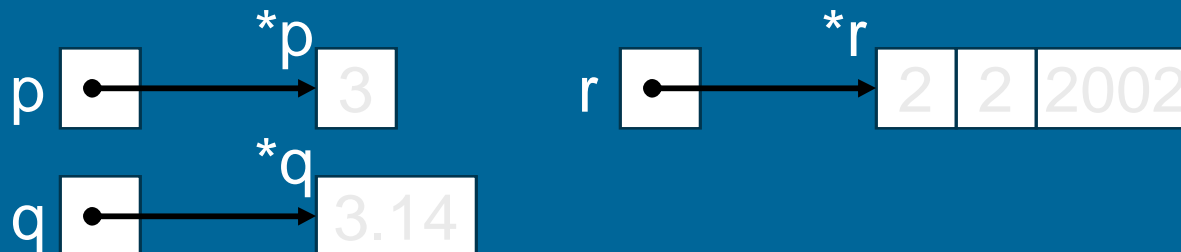


```
lesson12  
Auto  
Size of int is 4  
Size of int is 4  
Size of Employee is 44  
Size of john is 44  
Size of char is 1  
Size of short is 2  
Size of int is 4  
Size of long is 4  
Size of float is 4  
Size of double is 8  
Press any key to continue_
```


malloc()

Gunakan pointer dan function `malloc()` untuk membuat dynamic variables. `malloc()` mengalokasikan storage space.

```
int *p;  
double *q;  
Date *r;  
p = (int*) malloc(sizeof(int));  
q = (double*) malloc(sizeof(double));  
r = (Date*) malloc(sizeof(Date));
```



malloc(), closer look

```
int *p;  
p = (int*) malloc(sizeof(int));
```

sizeof(int) = the number of bytes occupied by an int

malloc() mengalokasikan storage dengan ukuran 4 byte (int)

p = **(int*)** malloc(...)
Karena p adalah pointer to int, **we have to cast** the RHS by (int*). You can think it as a hint to the C compiler that the allocated cell will be pointed by a pointer to integer.

malloc()

```
#include <stdlib.h>
int main () {
    int *p; int a=2;
    p = (int*) malloc(sizeof(int));
    *p = 4;
    *p += a;
    ...
    free(p);
    p = NULL;
}
```

Allocate a cell big enough for an int, dan mengarahkan pointer p untuk menunjuknya.

We create a dynamic variable using pointer and `malloc()`. Notice that the only name of the dynamic variable is `*p`. Therefore, we must access the variable through pointer.



free()

```
#include <stdlib.h>
int main () {
    int *p; int a=2;
    p = (int*) malloc(sizeof(int));
    *p = 4;
    *p += a;
    ...
    free(p);
    p = NULL;
}
```

Free the cell
pointed by p

When a dynamic variable is no longer used, we should free it. The compiler will not de-allocate the storage space automatically.

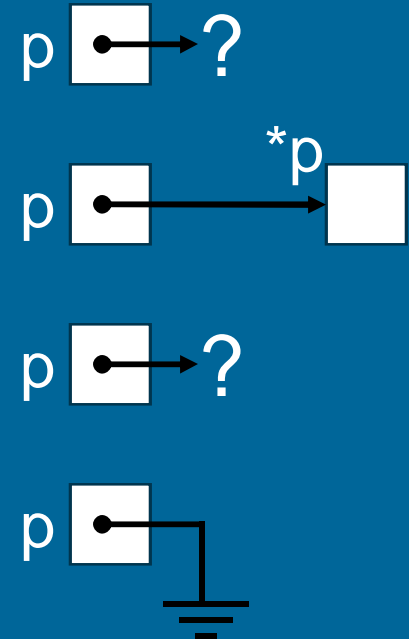


NULL

```
#include <stdlib.h>
int main () {
    int *p; int a=2;
    p = (int*) malloc(sizeof(int));
    *p = 4;
    *p += a;
    ...
    free(p);
    p = NULL;
}
```



After `free(p)`, `p` points to some rubbish. To indicate that `p` now points to nothing, it is a good practice to set `p = NULL`. `NULL` is a special pointer value.



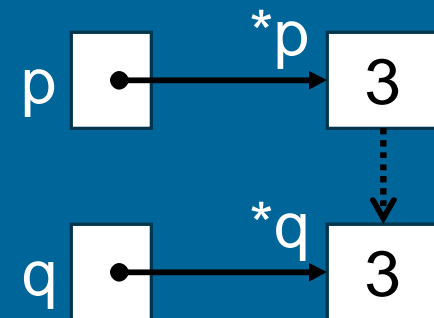
#include <stdlib.h>

```
#include <stdlib.h>
int main () {
    int *p; int a=2;
    p = (int*) malloc(sizeof(int));
    *p = 4;
    *p += a;
    ...
    free(p);
    p = NULL;
}
```

The function malloc(), free() and the special value NULL are defined in the header file **stdlib.h**. You should include it when you use dynamic variables.

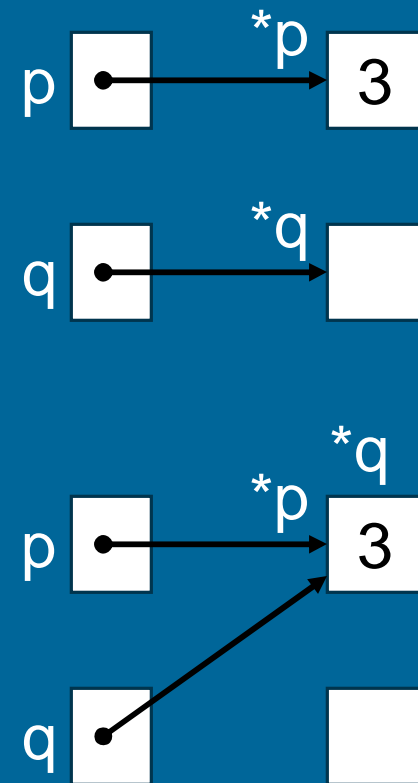
$$*q = *p$$

```
int *p, *q;  
p = (int*)malloc(sizeof(int));  
q = (int*)malloc(sizeof(int));  
*p = 3;  
*q = *p;
```



$q = p$

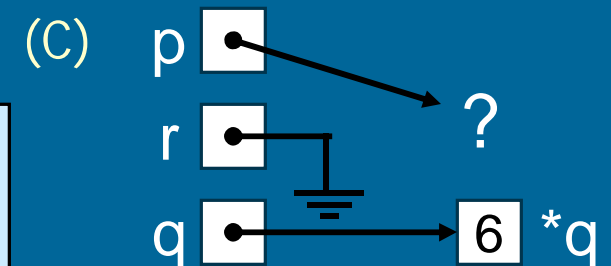
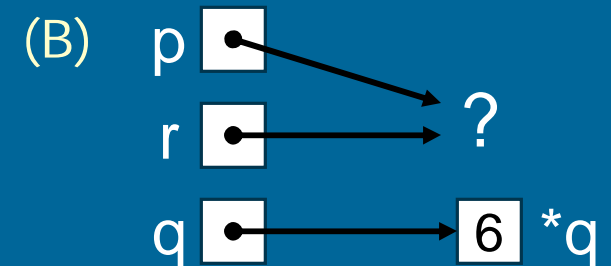
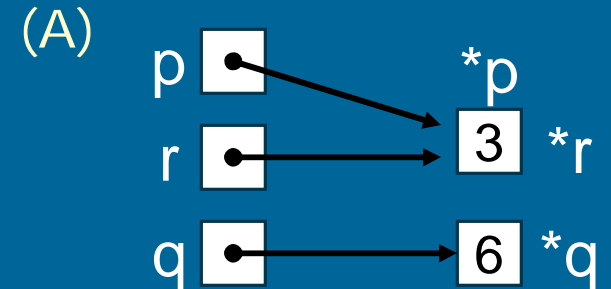
```
int *p, *q;  
p = (int*) malloc(sizeof(int));  
q = (int*) malloc(sizeof(int));  
*p = 3;  
q = p;
```



Example

```
#include <stdlib.h>
int main () {
    int *p, *q;
    int *r;
    p = (int*) malloc(sizeof(int));
    r = p;
    q = (int*) malloc(sizeof(int));
    *p = 3;
    *q = *p + *r; (A)
    free(r); (B)
    r = NULL; (C)
}
```

Remember, `free(r)` means 'free the dynamic variable pointed by the pointer `r`'. It does not change `r`. And after `free()`, `*r` and `*p` will be rubbish.



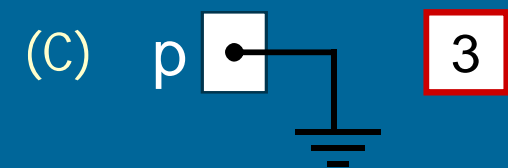
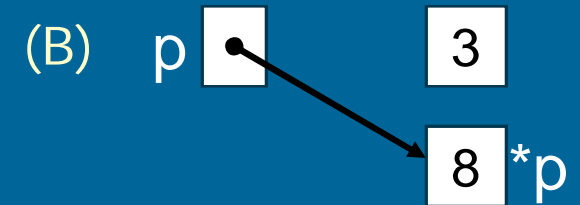
Exercise

```
#include <stdlib.h>
int main () {
    int *p, *q;
    int *r; int *s;
    p = (int*) malloc(sizeof(int));
    r = p; *p = 5;
    p = (int*) malloc(sizeof(int));
    q = p; s = r; *q = *s + 1;
    *r = *p * 2;
    (A)
    free(r); r = NULL;
    free(q); q = NULL;
    (B)
}
```

Q6) Use a diagram to illustrate the state of dynamic variables at the point A and B.

Memory Leak

```
#include <stdlib.h>
int main () {
    int *p, *q;
    p = (int*) malloc(sizeof(int));
    *p = 1+2; (A)
    q = p; or free(p);
    p = (int*) malloc(sizeof(int));
    scanf("%d", p); (B)
    free(p); p = NULL; (C)
}
```



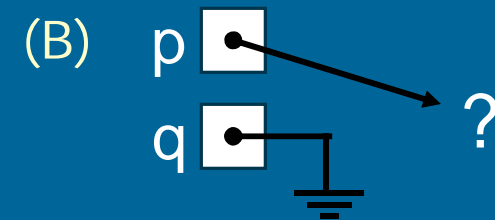
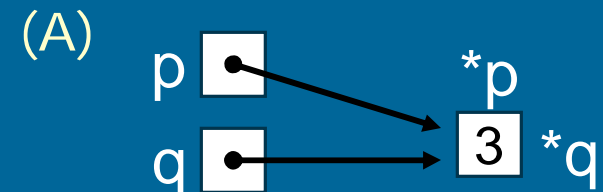
A dynamic variable must be accessed through a pointer. So before you change a pointer, (e.g. `p = q; p=(int*)malloc(...); p=NULL;`), beware that `*p` might not be accessible any longer.

Invalid Pointer De-reference

```
#include <stdlib.h>
int main () {
    int *p, *q;
    p = (int*) malloc(sizeof(int));
    q = p;
    *q = 3; (A)
    free(q);
    q = NULL; (B)
    printf("%d %d", *p, *q);
    *p = 999;
    *q = 999;
}
```



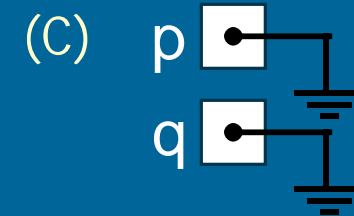
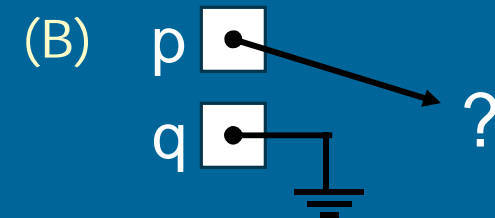
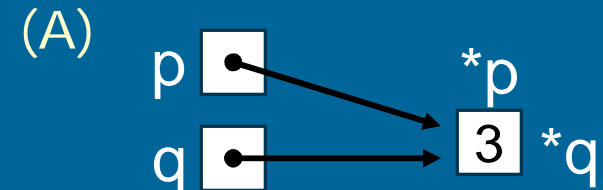
After free(q), the cell *p (and *q) disappears. Therefore, we can no longer use *p or *q.



Free unused cell once, and only once!

```
#include <stdlib.h>
int main () {
    int *p, *q;
    p = (int*) malloc(sizeof(int));
    q = p;
    *q = 3; (A)
    free(q);
    q = NULL; (B)
    free(p);
    p = NULL; (C)
}
```

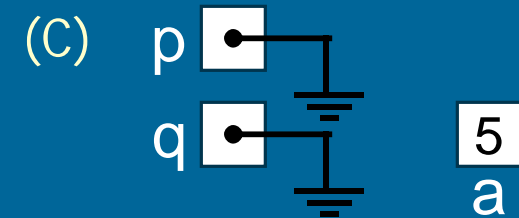
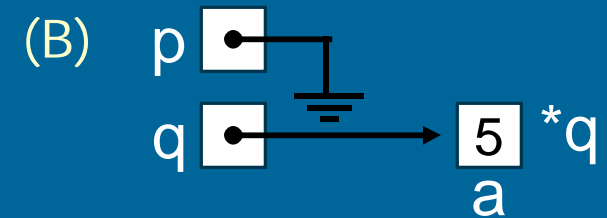
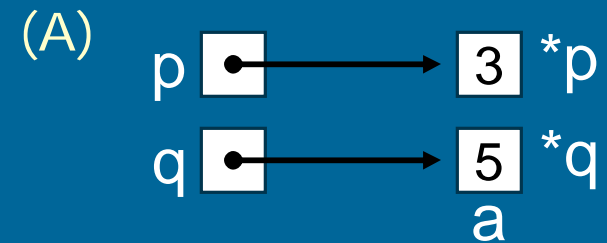
Free each unused cell once,
and only once. (NOT 'free
each pointer once')



Don't Free Static Variable

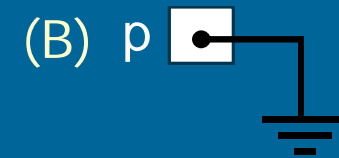
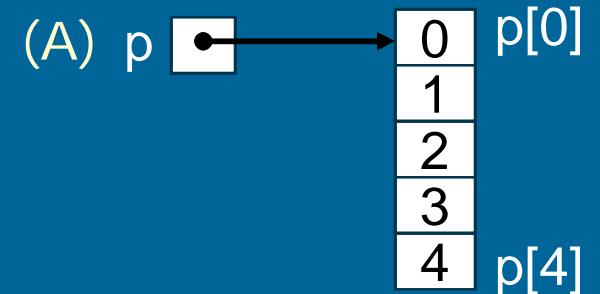
```
#include <stdlib.h>
int main () {
    int *p, *q;
    int a = 5;
    p = (int*) malloc(sizeof(int));
    q = &a;
    *p = 3; (A)
    free(p); p = NULL; (B)
    free(q);
    q = NULL; (C)
}
```

You only need to free dynamic variable. Never free static variable. Remember, free() what you malloc().



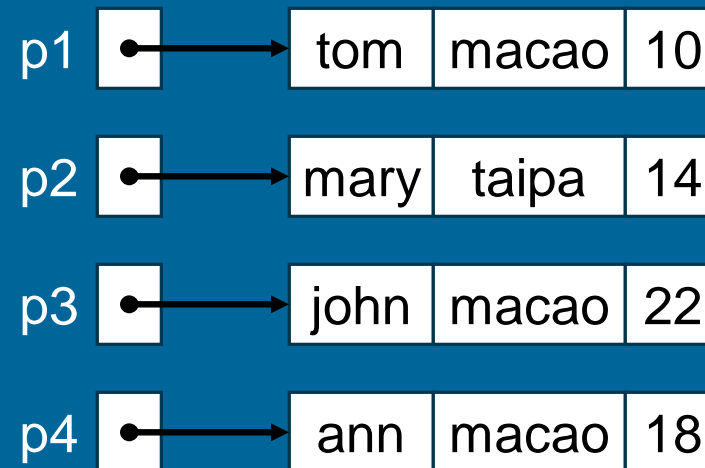
Dynamic Array

```
#include <stdlib.h>
int main () {
    int *p;
    int size=5;
    p = (int*) malloc(sizeof(int)*size);
    for (int i=0; i<size; i++) p[i] = i;
    (A)
    free(p);  p = NULL; (B)
}
```



You can also create an array as dynamic variable using malloc(). The array size need not be fixed at compile time.

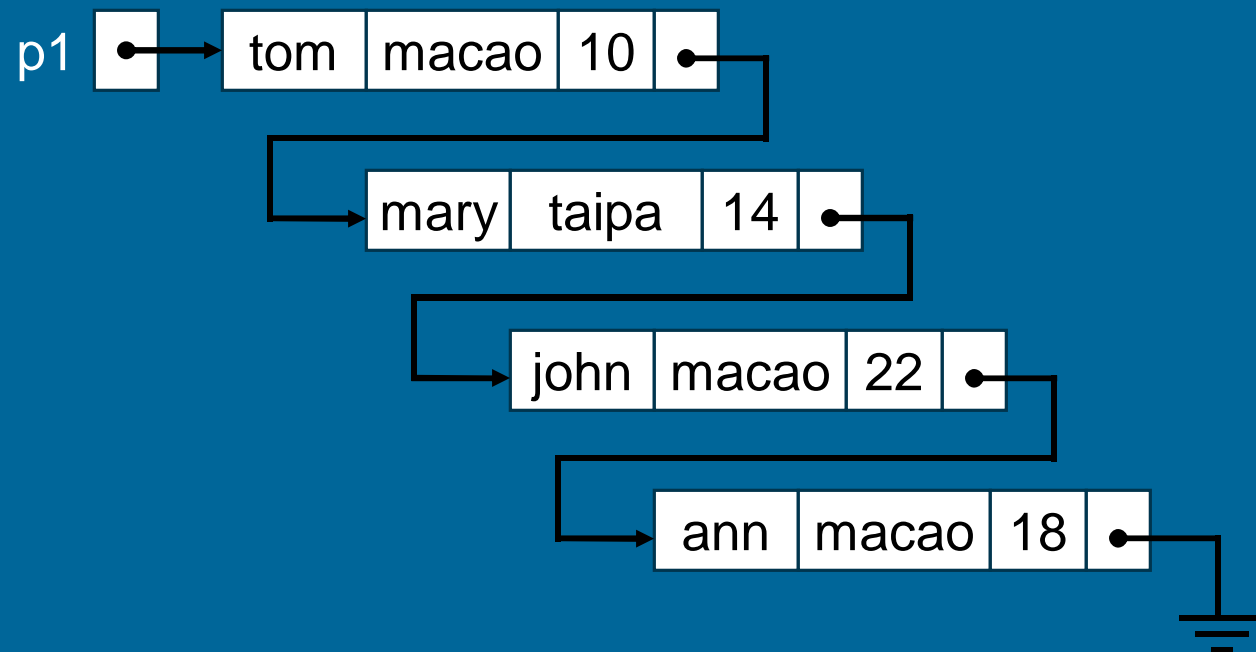
Linked List, Introduction



As we create more and more student records as dynamic variable, we need some way to organize our data. There are a lot of data structures.

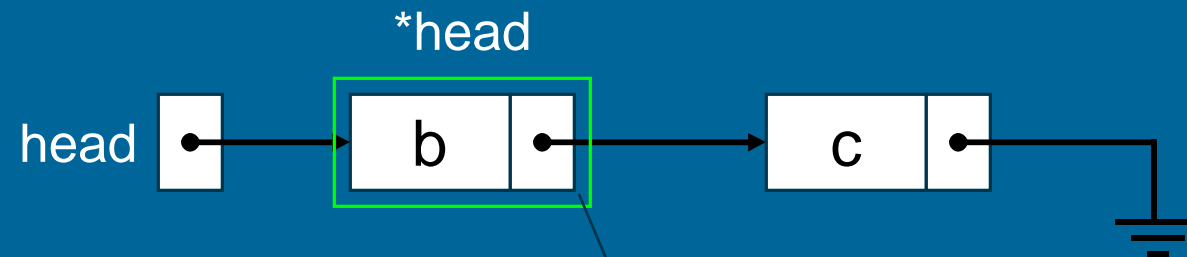
That's why there are two courses of Data Structure!

Linked List, Introduction



One way to do this is to linked all the student records together as a list with pointers.

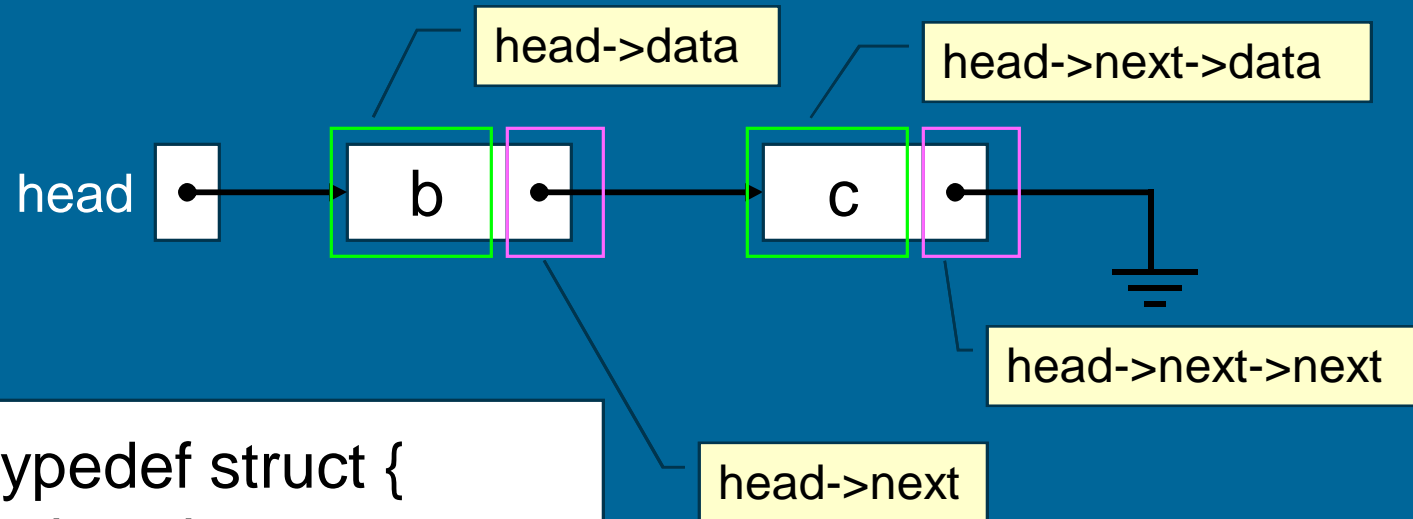
Linked List



```
typedef struct {  
    char data;  
    Node *next;  
} Node;  
  
Node *head;
```

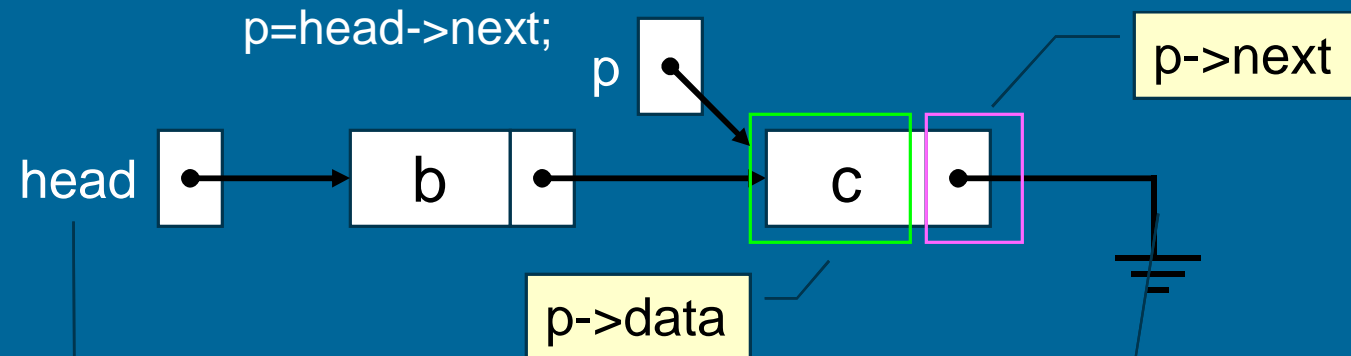
- Tiap simpul terdiri atas 2 bagian : data dan pointer *next*.
- Pointer *head* menunjuk ke simpul pertama dr linked list
- Pointer *head->next* menunjuk ke simpul kedua, dst.

Linked List



```
typedef struct {  
    char data;  
    Node *next;  
} Node;  
  
Node *head;
```


Linked List

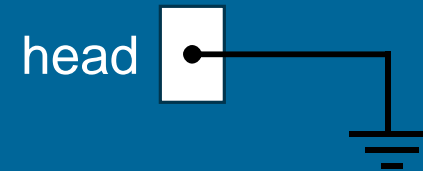


Pointer next dari
simpul terakhir =
NULL

Simpul pertama
ditunjuk oleh
sebuah pointer :
head

Empty linked list

- Linked list yang masih kosong ditunjukkan oleh pointer head yang bernilai NULL.
- Ingat, pointer head menunjuk ke simpul pertama dalam sebuah linked list.
- Jika list masih kosong, maka tidak terdapat simpul pertama.



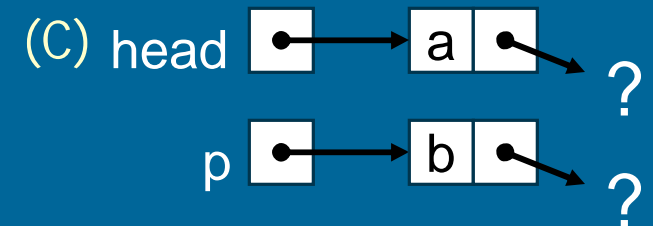
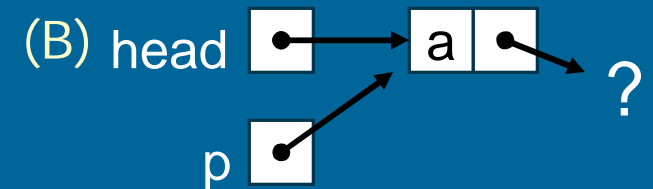
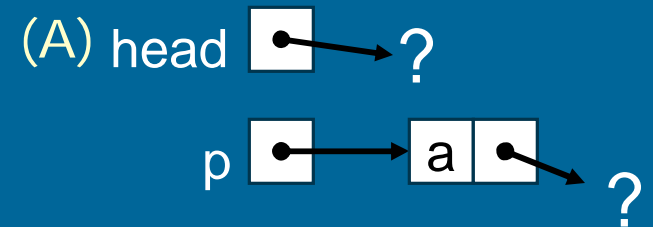
Utk singkatnya,
biasanya sel
pointernya tidak
digambar.



Example

```
#include <stdlib.h>
typedef struct {
    char data;
    Node *next;
} Node;

int main ( ) {
    Node *head;
    Node *p;
    p = (Node*) malloc(sizeof(Node));
    p->data = 'a'; (A)
    head = p; (B)
    p = (Node*) malloc(sizeof(Node));
    p->data = 'b'; (C)
    ...
}
```

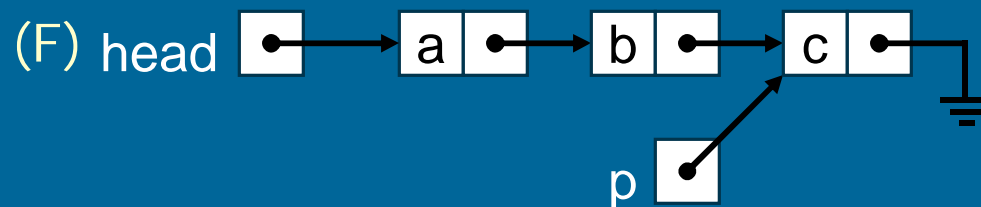
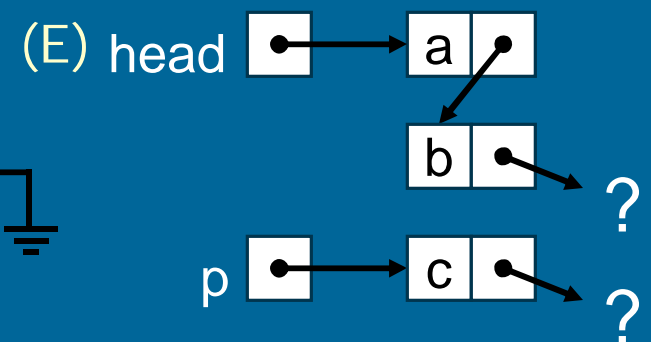
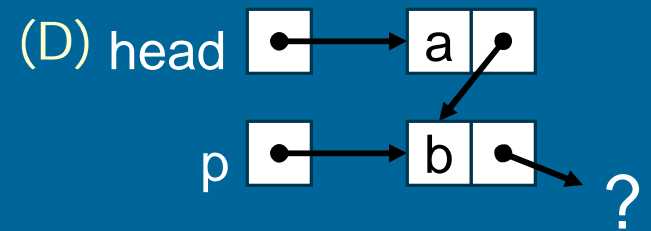
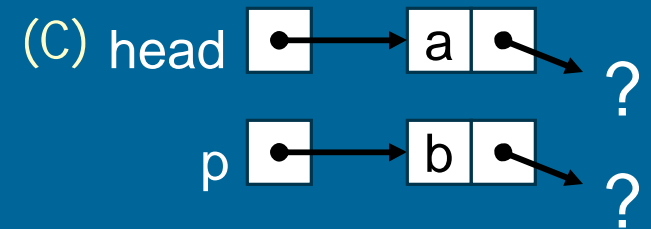


Example

```

... (C)
head->next = p; (D)
p = (Node*) malloc(sizeof(Node));
p->data = 'c'; (E)
head->next->next = p;
p->next = NULL; (F)
...
}

```



Exercise

Q7) Program di bawah ini membuat sebuah linked list berisi karakter 'a', 'b', dan 'c'. Trace program tsb dengan teliti. Gambarlah sebuah diagram untuk mengilustrasikan nilai dari dynamic variables dan pointer pada titik (A), (B) dan (C).

```
#include <stdlib.h>
typedef struct {
    char data;
    Node *next;
} Node;

...
```

Exercise

```
...
main ( ) {
    Node *head = NULL;
    Node *p;

    p = (Node*) malloc(sizeof(Node));
    p->data = 'a'; (A)
    p->next = (Node*) malloc(sizeof(Node));
    p->next->data = 'b';
    p->next->next = NULL; (B)
    head = (Node*) malloc(sizeof(Node));
    head->data = 'c';
    head->next = p; (C)
    ...
}
```


Basic Operations

```
typedef struct {  
    char data;  
    Node *next;  
} Node ;
```

```
typedef struct {  
    Node *head;  
} List;
```

```
void insert_list(List *L, char c);  
void del_list(List *L, char *c);  
void print_list(List L);
```

Diinginkan untuk menulis fungsi untuk memasukkan sebuah simpul ke dalam linked list, menghapus sebuah simpul dari linked list, dan mencetak seluruh simpul dalam linked list.

Sample usage

```
#include <stdio.h>
#include <stdlib.h>
```

```
typedef struct {
    char data;
    Node *next;
} Node ;
```

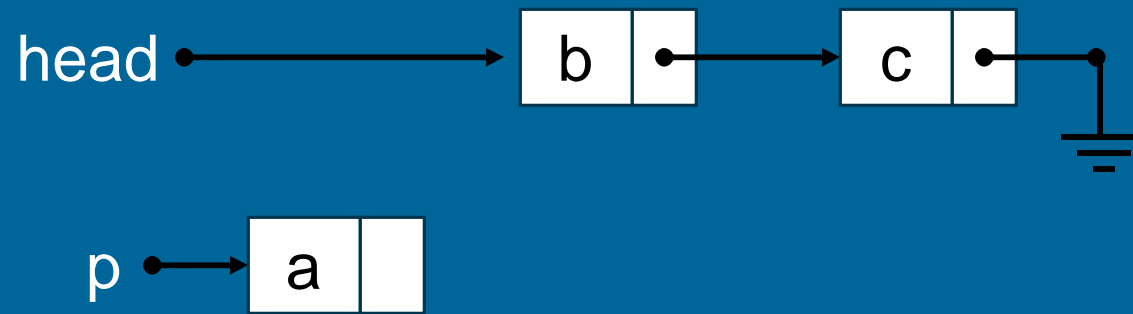
```
typedef struct {
    Node *head;
} List;
```

```
void insert_list(List *L, char c) { todo }
void del_list(List *L, char *c) { todo }
void print_list(List L) { todo }
```

```
int main () {
    char c;
    List L;   L.head = NULL;

    print_list(L);
    insert_list(&L, 'a'); insert_list(&L, 'b');
    print_list(L);
    del_list(&L, &c);
    printf("I just delete this %c\n", c);
    print_list(L);
    return 0;
}
```

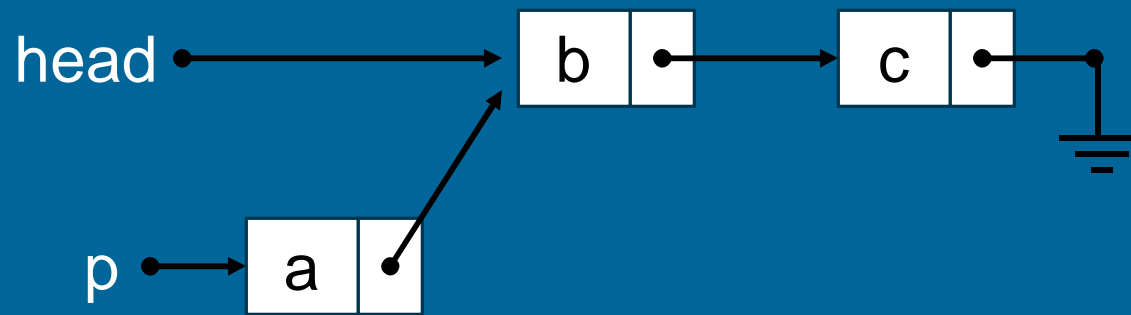
Adding a node, 1



Pertama, siapkan simpul pertama

```
Node *p;  
p = (Node*) malloc (sizeof(Node));  
p->data = 'a';
```

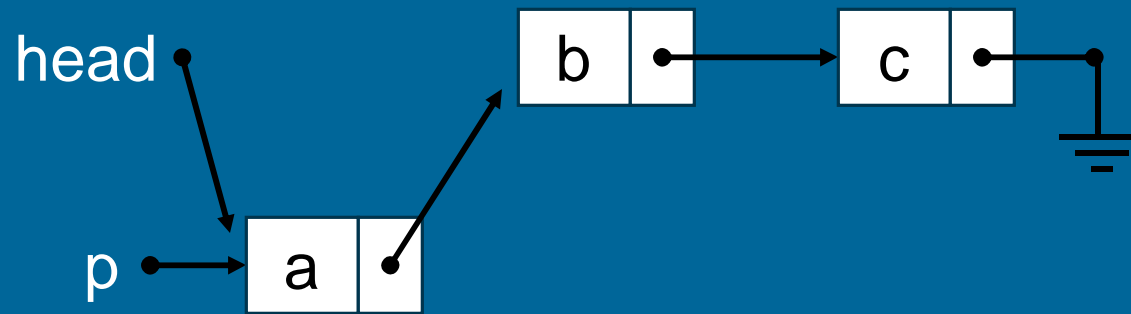
Adding a node, 2



Kedua, hubungkan simpul baru ke simpul pertama yang ditunjuk oleh head.

```
p->next = head;
```

Adding a node, 3

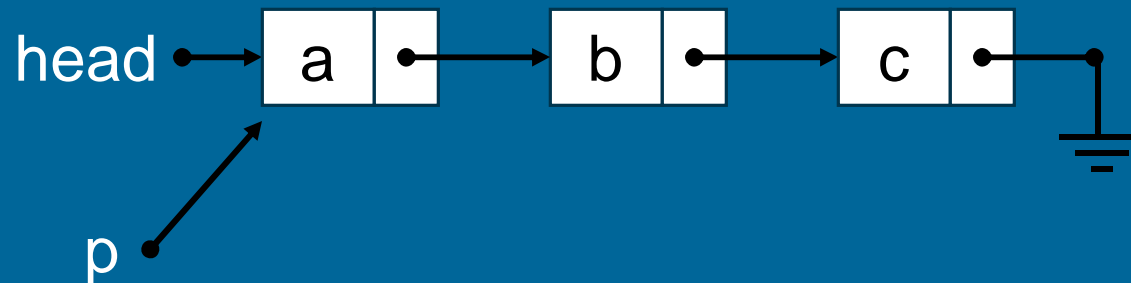


Terakhir, pointer head diupdate, sehingga menunjuk ke simpul pertama. Sekarang, simpul pertama berisi data 'a'

```
head = p;
```

Important: Check whether this function works for empty list.

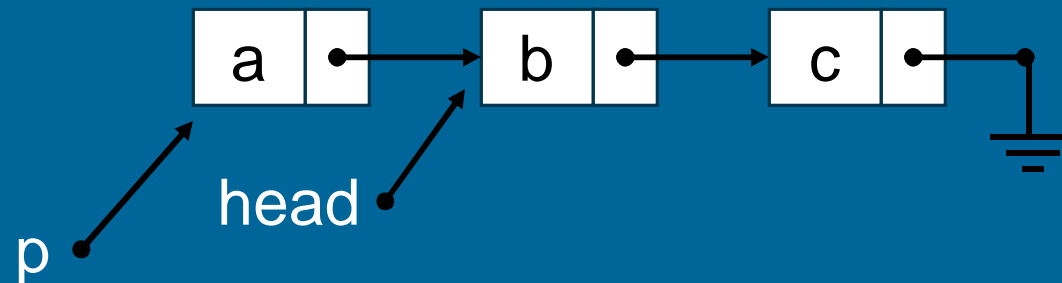
Deleting the first node, 1



Pertama, arahkan pointer bantuan p ke simpul pertama yang akan dihapus.

```
Node *p;  
p = head;
```

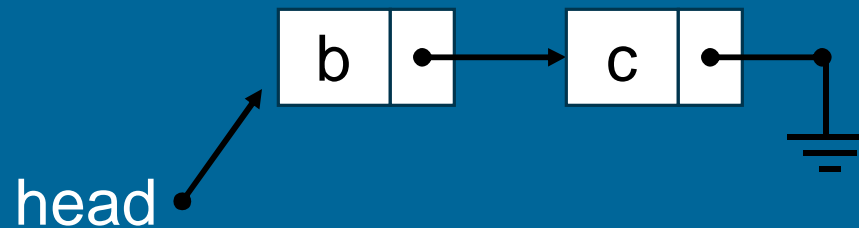

Deleting the first node, 2



Kedua, head pointer diupdate, sehingga menunjuk ke simpul kedua dari list yang asli.

```
head = p->next;
```

Deleting the first node, 3

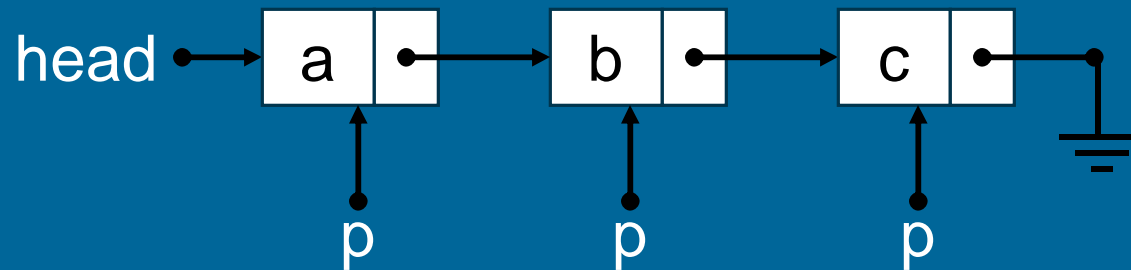


Terakhir, bebaskan lokasi yang ditunjuk oleh pointer p.

```
printf("%c", p->data);  
free(p);  
p = NULL;
```

Important: Make sure your function can handle list with a single node.

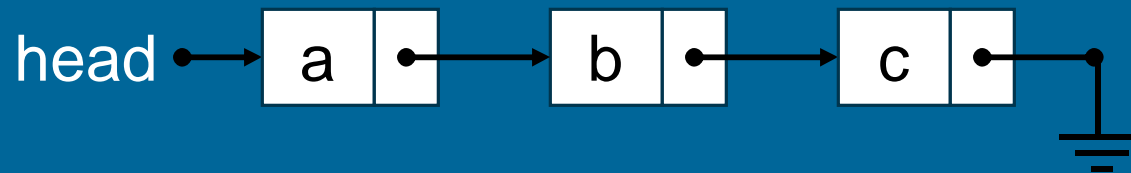
Traversing a linked list, 1



Pointer p diarahkan ke node ybs. Setelah diproses, p diupdate spy menunjuk ke node berikutnya.

```
p = p->next;
```

Traversing a linked list, 2



```
p = head;
while (p!=NULL) {
    printf("%c", p->data);
    p = p->next;
}
```

Important: Make sure your code can handle empty list.

Diawali dari simpul pertama (yg ditunjuk oleh pointer head). Loop sampai p menunjuk ke NULL.

```
p = head;
while (p) {
    ...
    p = p->next;
}
```

Program Listing, 1

```
#include <stdio.h>
#include <stdlib.h>
```

```
struct Node {
    char data;
    Node *next;
};
```

```
struct List {
    Node *head;
};
```

Put the following
three functions here.

```
int main () {
    char c;
    List L;
    L.head = NULL;
    printf("\n\n");
    print_list(L);
    insert_list(&L, 'a');
    insert_list(&L, 'b');
    print_list(L);
    del_list(&L, &c);
    printf("I just delete this %c\n", c);
    print_list(L);
    return 0;
}
```

Program Listing, 2

```
/* insert the char c to the head of the linked list L */  
/* !! no error checking for malloc failure */  
void insert_list (List *L, char c) {  
    Node *p;  
    p = (Node*) malloc(sizeof(Node));  
    p->data = c;  
    p->next = L->head;  
    L->head = p;  
}
```


Program Listing, 3

```
/* delete the first node of the linked list
 * and return the char stored there as *c
 * !! no error checking for deleting from empty list
 */
void del_list (List *L, char *c) {
    Node *p;
    p = L->head;
    L->head = p->next;
    *c = p->data;
    free(p);
    p = NULL;
}
```

Program Listing, 4

```
/* prints the content of the linked list L */  
void print_list (List L) {  
    Node *p;  
    p = L.head;  
    printf("The list contains [ ");  
    while (p!=NULL) {  
        printf("%c ", p->data);  
        p = p->next;  
    }  
    printf("]\n");  
}
```

Exercise

Q8) When `malloc()` cannot find enough memory, it returns `NULL`. Add checking for this error in the function `insert_list()`. If there is not enough memory, display an error message, abort the insertion of data and return 0. Otherwise, return 1 after successful insertion.

```
int insert_list (List *L, char c);
```

Exercise

Q9) User cannot delete a node from an empty linked list in `del_list ()`. Modify the function to display an error message, abort deletion and return 0 when the linked list is empty. Return 1 if deletion is successful and return the deleted character through the parameter `c`.

```
int del_list (List *L, char *c);
```

Q10) Write a function to count and return the number of nodes in a linked list.

```
int count_list (List L);
```

Modules

```
/* hello.cpp */  
#include <stdio.h>  
void greet (char *s) {  
    printf("Hello %s\n", s);  
}  
void goodbye ( ) {  
    printf("Bye bye");  
}
```

```
int main ( ) {  
    greet("Peter");  
    goodbye();  
    return 0;  
}
```

Sometimes, your program might be getting too long to fit in a single file. We can use modules to extract some functions to a separate file.

Need of Header

```
/* hello.cpp */
```

```
int main () {  
    greet("Peter");  
    goodbye();  
    return 0;  
}
```

Er.. What
is greet()?

```
/* greet.cpp */  
#include <stdio.h>
```

```
void greet (char *s) {  
    printf("Hello %s\n", s);  
}  
void goodbye ( ) {  
    printf("Bye bye");  
}
```

Splitting the program into two files raises two problems. First, how does the main program know the extracted functions `greet()` and `goodbye ()`?

Need of Project

```
/* hello.cpp */
```

```
int main () {  
    greet("Peter");  
    goodbye();  
    return 0;  
}
```

+

```
/* greet.cpp */  
#include <stdio.h>
```

```
void greet (char *s) {  
    printf("Hello %s\n", s);  
}  
void goodbye ( ) {  
    printf("Bye bye");  
}
```

=

```
hello.exe
```

Second, how do we tell the compiler that our program consists of more than one files, namely hello.cpp and greet.cpp?

Header file

```
/* hello.cpp */  
#include "greet.h"  
int main () {  
    greet("Peter");  
    goodbye();  
    return 0;  
}
```

A solution to the first problem is to introduce the extracted functions in a header file. We then include the file in the main program.

```
/* greet.h */  
extern void greet (char *s);  
extern void goodbye();
```

```
/* greet.cpp */  
#include <stdio.h>  
#include "greet.h"  
void greet (char *s) {  
    printf("Hello %s\n", s);  
}  
void goodbye ( ) {  
    printf("Bye bye");  
}
```

#include

```
/* hello.cpp */  
#include "greet.h"  
int main () {  
    greet("Peter");  
    goodbye();  
    return 0;  
}
```

```
/* greet.h */  
extern void greet (char *s);  
extern void goodbye();
```

```
/* hello.cpp */  
extern void greet (char *s);  
extern void goodbye();  
int main () {  
    greet("Peter");  
    goodbye();  
    return 0;  
}
```

The included statements list the signature of the functions and tell the main program that the functions are **external**,

#include

Although we don't need to do so, it is a good practice to include the header in the module. It helps to check that the signature of functions in the header file is the same as those in the module.

```
/* greet.h */  
extern void greet (char *s);  
extern void goodbye();
```

```
/* greet.cpp */  
#include <stdio.h>  
#include "greet.h"  
void greet (char *s) {  
    printf("Hello %s\n", s);  
}  
void goodbye ( ) {  
    printf("Bye bye");  
}
```

Project

```
/* hello.cpp */
#include "greet.h"
int main () {
    greet("Peter");
    goodbye();
    return 0;
}
```

```
/* greet.h */
extern void greet (char *s);
extern void goodbye();
```

```
/* greet.cpp */
#include <stdio.h>
#include "greet.h"
void greet (char *s) {
    printf("Hello %s\n", s);
}
void goodbye ( ) {
    printf("Bye bye");
}
```

To solve the second problem, we use project in Turbo C. We tell the compiler that these two C files form a single program by putting them in a project. Detail will be given in demo in class.

Module for Reuse

```
/* hello2.cpp */
#include "greet.h"
int main () {
    greet("Mary");
    greet("John");
    goodbye();
    return 0;
}
```

```
/* greet.h */
extern void greet (char *s);
extern void goodbye();
```

```
/* greet.cpp */
#include <stdio.h>
#include "greet.h"
void greet (char *s) {
    printf("Hello %s\n", s);
}
void goodbye ( ) {
    printf("Bye bye");
}
```

In addition to keeping each source file shorter, modules also help to reuse functions. `stdio.h`, `stdlib.h`, `string.h`, etc.. are standard libraries provided by the C compiler.