Programming in C

Lecture 4

Thanks to Arjan van Genderen, TU-Delft, for the slides

Announcements

- Diagnostic test Oct 7th (focus on theory of last two lectures including labs).
- Lab assignments:
  - Advice: finish exercises of chapter 4 (lab guide) before en of week 40
  - All assignments must be completed before Oct 24th.

Lecture 4

- Composite data types:
  - Structures
  - Unions
- Data structures:
  - Lists
  - Trees
  - Graphs
- Input/Output and the Operating System

Book chapters 9 until 11
(NOT: 9.8 – 9.10, 10.6, 11.6-11.9, 11.11-11.12, 11.14-11.19)

Structures

Structures

Structures: A means to combine different variables:

```
struct structname {
   type1 membername1;
   type2 membername2;
   ...
};
```

Example:

```
struct student {
   char *name;
   int id;
};

struct student person1, person2;

person1.name = "J. Jansen";
person1.id = 4012345;

person2 = person1;
```


Structure definition and declaration

Definition and declaration together:

```
struct student {
   char *name;
   int id;
} person1, person2;
```

It is more convenient to separate definition and declaration:

```
typedef struct {
   char *name;
   int id;
} student;
```

Or:

```
struct student {
   char *name;
   int id;
};
typedef struct student student;
```

```
student person1, person2;

student allyear1[150];

allyear1[3].name = "P. Pietersen";
allyear1[3].id = 4099999;
```

Pointers to structures

```
student * p1;

p1 = calloc (1, sizeof (student));
(*p1).id = 4099999; (parentheses are required, else *(p1.id))
```

The member access operator → is often used:

```
p1 -> id = 4099999;    /* p1 -> id is the same as (*p1).id */
```

Example:

```
typedef struct {
   double re;
   double im;
} complex;

void add (complex *a, complex *b, complex *c)
{
   a -> re = b -> re + c -> re;
   a -> im = b -> im + c -> im;
}
```


Hierarchical structures

```
struct dept {
   char dept_name[25];
   int dept_no;
};


typedef struct {
   char                 name[25];
   struct dept          department;
   struct home_address  *a_ptr;
   double               salary;
   ...
} employee_data;


employee_data e;

e.department.dept_no = 3;
```

struct home_address need not necessarily to be know here since only space is reserved for the pointer in this structure.

Structures as function argument

```
employee_data update (employee_data e)
{
   .....
   printf ("Input the department number: ");
   scanf ("%d", &n);
   e.department.dept_no = n;
   .....
   return e;
}
```

Example function call:

```
employee_data e1;
e1 = update(e1);
```

Disadvantage: whole structure is copied twice:
- when the function is called (call by value), and
- at the end of the function

Structures as function argument – Improved version

```
void update (employee_data * p)
{
   .....
   printf ("Input the department number: ");
   scanf ("%d", &n);
   p -> department.dept_no = n;
   .....
}
```

Example function call:

```
employee_data e1;
update(&e1);
```

No structure is copied.

Hence, for (large) structures: pass the pointer value !

Unions

A union is a structure in which the members share memory space.

```
union int_or_float {
   int     i;
   float   f;
};

int main(void)
{
   union int_or_float  n;

   n.i = 4444;
   printf("i: %10d     f: %16.10e\n", n.i, n.f);
   n.f = 4444.0;
   printf("i: %10d     f: %16.10e\n", n.i, n.f);
   return 0;
}
```

i and f use same memory space.

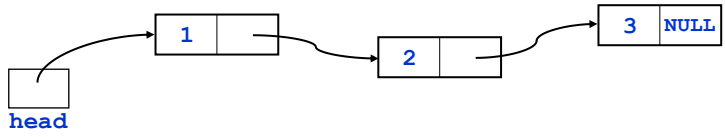
Output:

```
i:        4444     f: 6,227370375e-41
i: 1166729216      f: 4.4440000000e+03
```

Remember how members are used !

Lists

Linear linked list



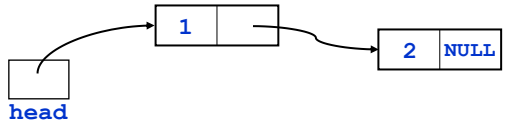

```
struct list {
   int         data;
   struct list * next;
};

struct list * head;
```

Lists are used when the number of elements is variable.
Possible operations, e.g.:

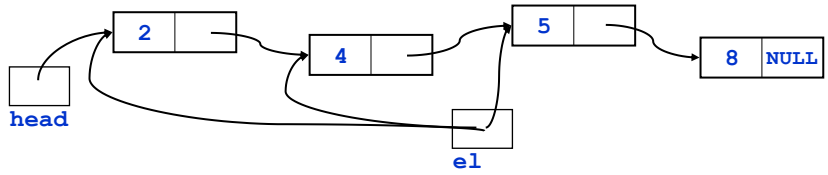
- search
- add
- remove

Linear linked list - constructing

```
struct list {
   int         data;
   struct list * next;
};

struct list * head;

head = malloc (sizeof (struct list));
head -> data = 1;
head -> next = NULL;

head -> next = malloc (sizeof (struct list));
head -> next -> data = 2;
head -> next -> next = NULL;
```

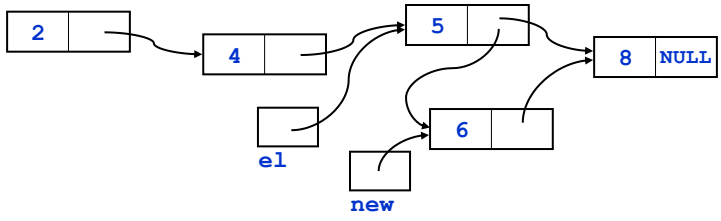


Linear linked list - search



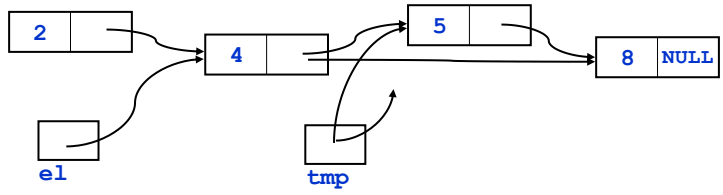

```
struct list * head;

/* search for element that contains data value 5 */

struct list * el;

el = head;
while (el != NULL && el -> data != 5)
   el = el -> next;
```


Linear linked list – add




```
/* add element new after element pointed to by el */

new -> next = el -> next;

el -> next = new;
```

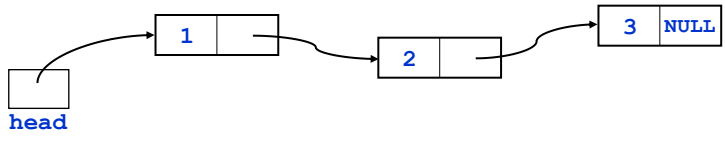
Linear linked list - remove




```
/* remove element after element pointed to by el */

struct list * tmp;
tmp = el -> next;
el -> next = el -> next -> next;
free (tmp);
```

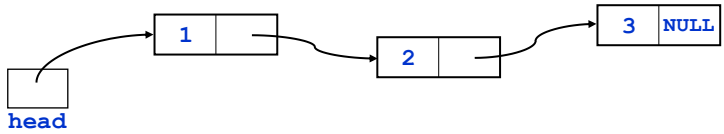
Release the used memory !

Print the list (while)




```
void print_list(struct list *list)
{
    while (list!=NULL)
    {
        printf("Value is %d\n",list->data);
        list=list->next;
    }
    printf("Printed all elements of the list\n");
}

print_list(head);
```

Print the list (recursive)



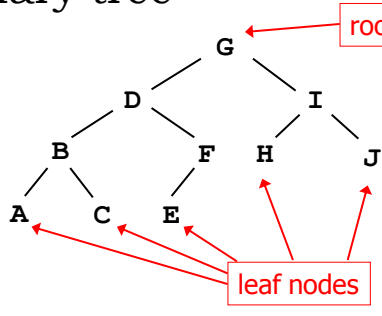

```
void print_list(struct list *list)
{
    if (list==NULL)
        printf("Printed all elements of the list\n");
    else
    {
        printf("Value is %d\n",list->data);
        print_list(list->next); /* recursive! */
    }
}

print_list(head);
```


List variants

- Double linked list
  - In addition to *next pointer* also *previous pointer*
- Stack
  - Only add and remove element in front of list
- Queue
  - The list has a *head pointer* and a *tail pointer*
  - Elements are added at the front of the list and removed from the tail (FIFO = First-In, First-Out)

Trees

Binary tree



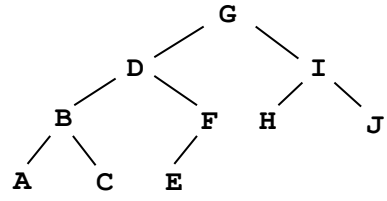
- In a binary tree an element has 0, 1 or 2 'children'
- Sorted binary tree:
  - Left sub tree:
    - elements are 'smaller'
  - Right sub tree:
    - elements are 'larger'

```
struct node {
   char        data;
   struct node * left;
   struct node * right;
};
```

For a lead node both are NULL

Advantage of a sorted binary tree over a linear list is that elements are normally reached, on average, in a logarithmic number of link traversals.

Binary tree traversal: inorder

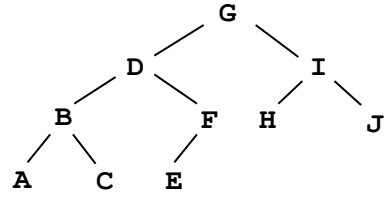



```
void inorder (struct node *n)
   if (n != NULL) {
      inorder (n -> left);
      printf ("%c ", n -> data);
      inorder (n -> right);
   }
};
```

Output:

A B C D E F G H I J

Binary tree traversal: preorder

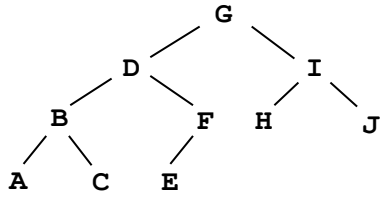



```
void preorder (struct node *n)
   if (n != NULL) {
      printf ("%c ", n -> data);
      preorder (n -> left);
      preorder (n -> right);
   }
};
```

Output:

G D B A C F E I H J

Binary tree traversal: postorder



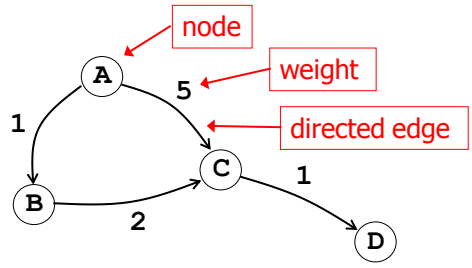

```
void postorder (struct node *n)
   if (n != NULL) {
      postorder (n -> left);
      postorder (n -> right);
      printf ("%c ", n -> data);
   }
};
```

Output:

A C B E F D H J I G

Graphs

Graphs



Example: search for a route in a map (cities and connecting roads (with length))

```
struct city {
   char        * name;
   struct road * roads;
};
```

```
struct road {
   struct city * origin;
   struct city * destination;
   unsigned    length;
   struct road * next;
};
```

In this example:
node → city
edge → road
weight → length

Each city has a linked list with the 'roads' that leave that city.

Input/Output and the Operating System

printf and scanf (slide from lecture 1)

```
#include <stdio.h>

int main(void)
{
    char     c1, c2, c3;
    int      i;
    float    x;
    double   y;
    int n;
    printf("%s\n%s",
           "Input three characters",
           "an int, a float, and a double:  ");
    n=scanf("%c%c%c%d%f%lf", &c1, &c2, &c3, &i, &x, &y);
    printf("Read %d variables.\n",n);
    printf("Here is the data that you typed in:\n");

    printf("%3c%3c%3c%5d%17e%17e\n\n", c1, c2, c3, i, x, y);
    return 0;
}
```

Input and output:

```
Input three characters
an int, a float, and a double: ABC  3  55  77.7
Read 6 variables.
Here is the data that you typed in:
  A  B  C    3     5.500000e+01      7.770000e+01
```


scanf/fscanf and printf/fprintf

```
scanf()

-Reads from the stdin (keyboard)

-Returns the number of variables read. E.g.

  n=scanf("%c%c%c%d%f%lf", &c1, &c2, &c3, &i, &x, &y);

  You expect that n is 6.

scanf("%d",&i);

is equal to:

fscanf(stdin,"%d",&i);


printf("funny example");

is equal to:

fprintf(stdout,"funny example");
```

Input/output with files

```
#include <stdio.h>

int main(int argc, char *argv[])
{
   FILE  *ifp, *ofp;
   int   a, sum;

   if (argc != 3) {
      printf ("Usage: %s inputfile outputfile\n", argv[0]);
      exit(1);
   }
   ifp = fopen(argv[1], "r");      /* open for reading */
   ofp = fopen(argv[2], "w");      /* open for writing */

   sum=0;
   while (fscanf (ifp, "%d", &a) == 1)
      sum += a;
   fprintf (ofp, "The sum is %d.\n", sum);

   fclose(ifp);
   fclose(ofp);
   return 0;
}
```

FILE is defined in stdio.h

When a file cannot be opened NULL is returned

close FILE

System commands

With the library function `system()`

windows/linux programs are executed from the C program

```
#include <stdio.h>
#include <stdlib.h>
int main (void)
{
    system ("dir");     /* execute command 'dir' */
    return 0;
}
```

```
...
24-09-2013  21:06    <DIR>          .
24-09-2013  21:06    <DIR>          ..
24-09-2013  21:06                 0 out.txt
24-09-2013  21:04           137.738 system.exe
               2 File(s)    137.738 bytes
               2 Dir(s)  106.802.688.000 bytes free
```

Summary

- Composite data types:
  - Structures
  - Unions
- Data structures:
  - Lists
  - Trees
  - Graphs
- Input/Output and the Operating System

Arduino project on Thursday at 13.45

Install the software on your laptop

http://arduino.cc/en/Guide/Windows