EE1400: Programmeren in C

BSc. EE, 1e jaar, 2012-2013, 4e college

Arjan van Genderen, Computer Engineering

11-12-2012

Mededelingen

- Voortgangstoets: Woensdagmiddag 19 dec.
  Stof: alles, met nadruk op laatste 2 colleges.
- Tentamen: Vrijdagochtend 1 feb. 2013
  Zorg dat je de hoofdlijnen (zie slides) begrijpt en vlot kunt toepassen. Details kun je altijd nazoeken in het C boek (boek en slides mogen erbij gehouden worden !).
- Om vak af te ronden moet practicum afgerond zijn.
  Practicumopdrachten kunnen ingeleverd worden tot vrijdag 28 dec. 2012, 24.00 uur. Voorkom vertraging: lees opdracht goed, test je programma goed en hou je aan de inleverregels.

Hoorcollege 4

- Samengestelde datatypen:
  - Structures
  - Unions
- Datastructuren:
  - Lists
  - Trees
  - Graphs
- Input/Output en het Operating System

Corresponderende stof in boek: Hoofdstuk 9 t/m 11
(NIET: 9.8 – 9.10, 10.6, 11.6-11.9, 11.11-11.12, 11.14-11.19)

Structures

Structures

Een middel om aggregaties van verschillende variabelen te vormen:

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

Bijvoorbeeld:

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

struct student person1, person2;

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

person2 = person1;
```

Structure definitie en declaratie

Definitie en declaratie ineen:

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

Maar handiger is definitie en declaratie te scheiden:

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

Of:

```
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 naar structures

```
student * p1;

p1 = calloc (1, sizeof (student));
(*p1).id = 4099999;
```

Dat laatste kan handiger met de -> constructie:

```
p1 -> id = 4099999;    /* p1 -> id == (*p1).id */
```

Voorbeeld:

```
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;
}
```

Hierarchische 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 hoeft hier nog niet perse bekend te zijn aan de compiler omdat alleen ruimte voor pointer wordt gereserveerd

Structures als functie argument

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

Aanroep:

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

Nadeel: complete structure wordt 2 maal gekopiëerd:

- bij begin van de functie (call by value)
- aan het einde van de functie

Structures als functie argument - Verbeterde versie

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

Aanroep:

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

Nu worden geen structures gekopiëerd.

Dus bij (grote) structures: pointer doorgeven !

Unions

Een union is een structure waarbij de members geheugenruimte delen.

```
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 en f worden opgeslagen op hetzelfde geheugenadres

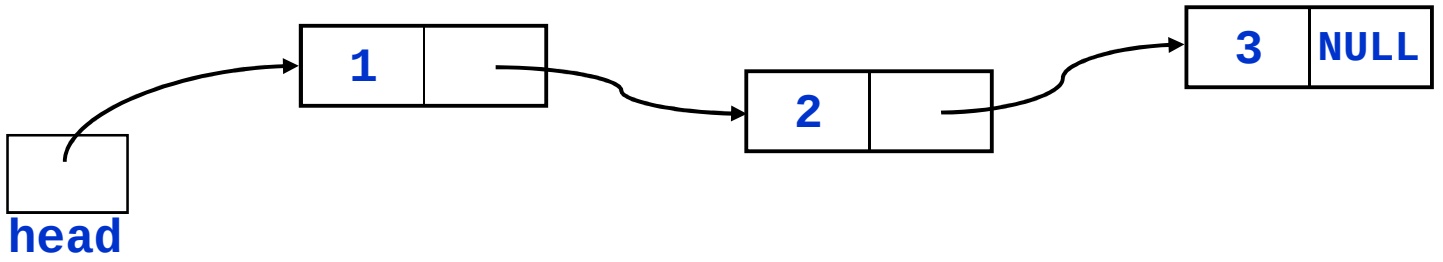
Output:

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

houdt dus bij welk member actueel is !

Lists

Linear linked list



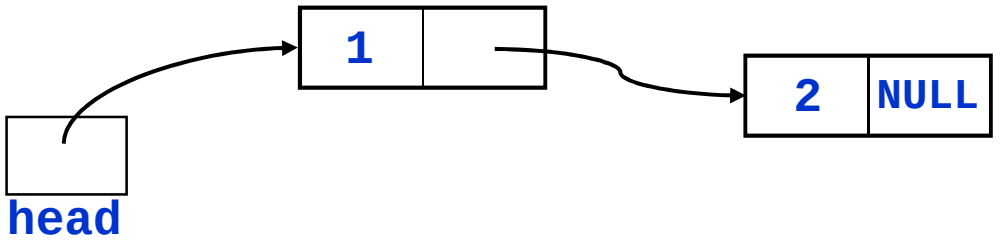

```
struct list {
   int          data;
   struct list * next;
};
struct list * head;
```

Lists worden gebruikt wanneer het aantal elementen variabel is.
Mogelijke operaties o.a.:

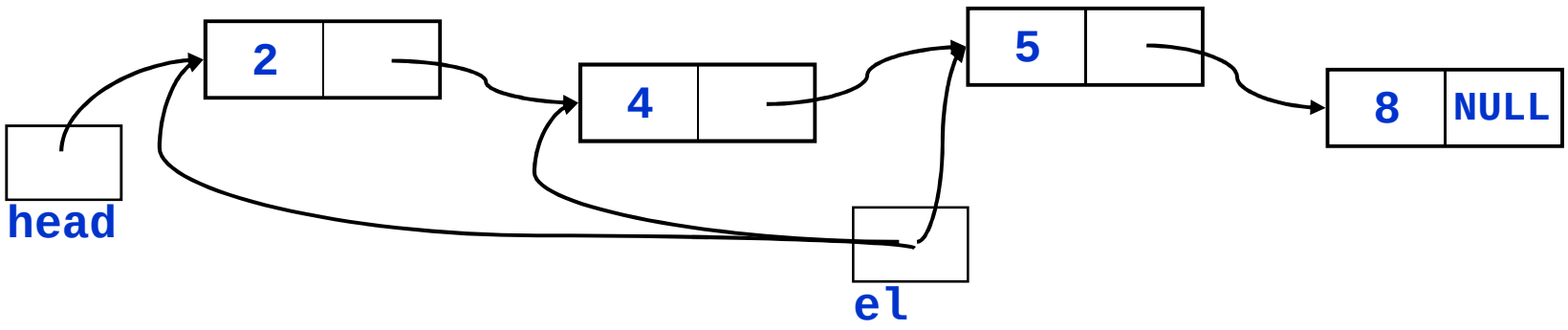
- zoeken
- toevoegen
- verwijderen

Linear linked list - opbouwen

```
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;
```

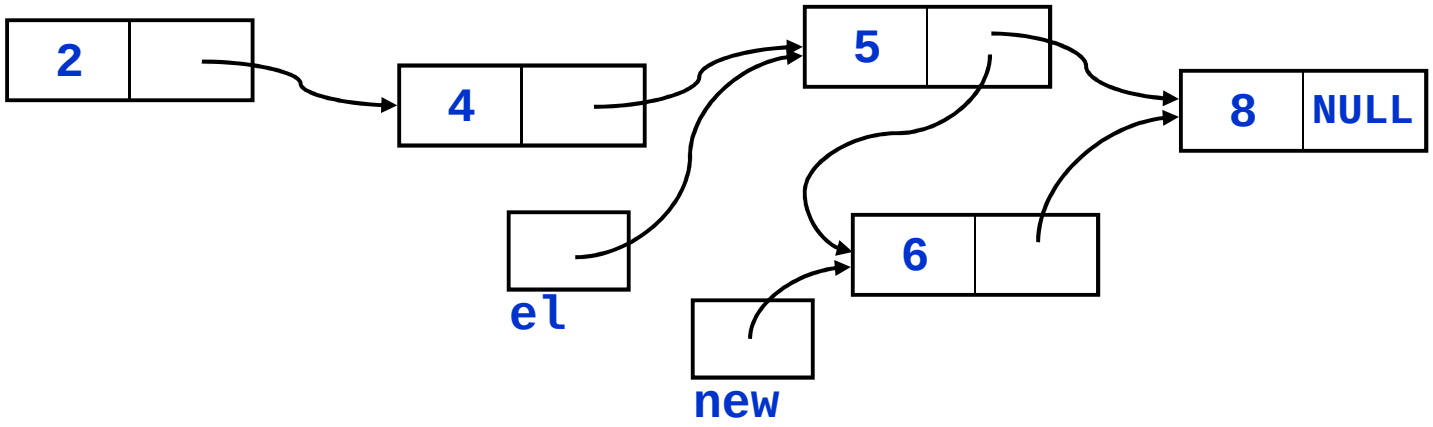


Linear linked list - zoeken



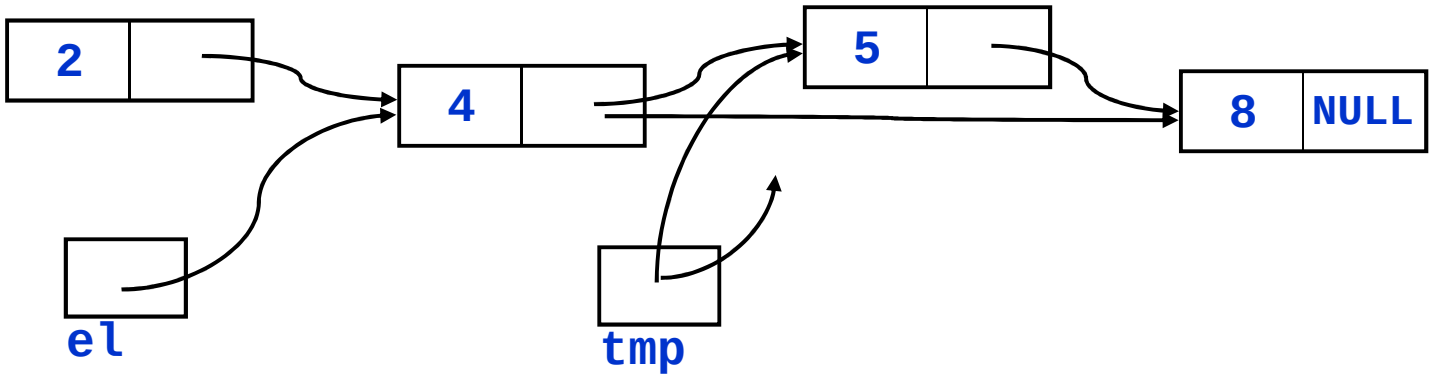

```
struct list * head;

/* 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 - toevoegen




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

new -> next = el -> next;

el -> next = new;
```

Linear linked list - verwijderen




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

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

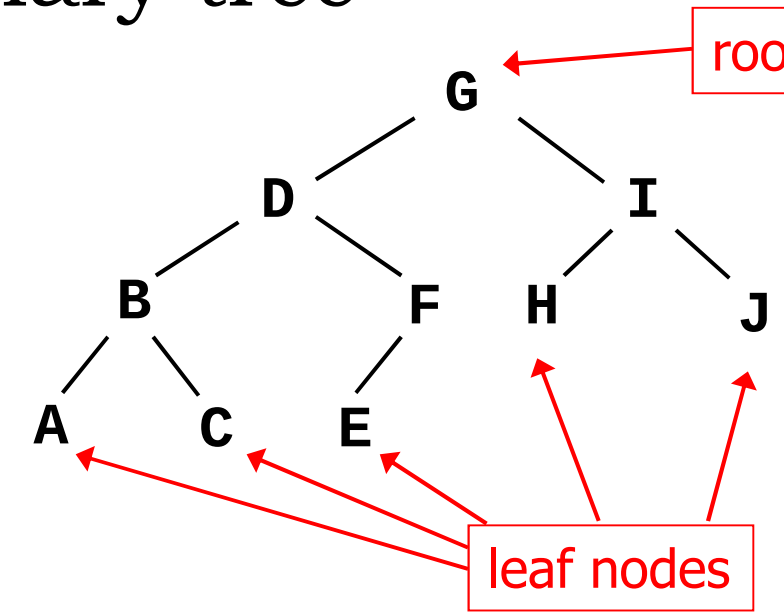
tmp wijst nu naar iets
dat niet meer bestaat !

List varianten

- Double linked list
  - Naast next pointer ook previous pointer in structure
- Stack
  - Alleen elementen toevoegen en verwijderen vooraan
- Queue
  - Naast head van list wordt ook tail van list bijgehouden
  - Elementen worden alleen toevoegd vooraan en verwijderd achteraan (FIFO = First-In, First-Out)

Trees

Binary tree



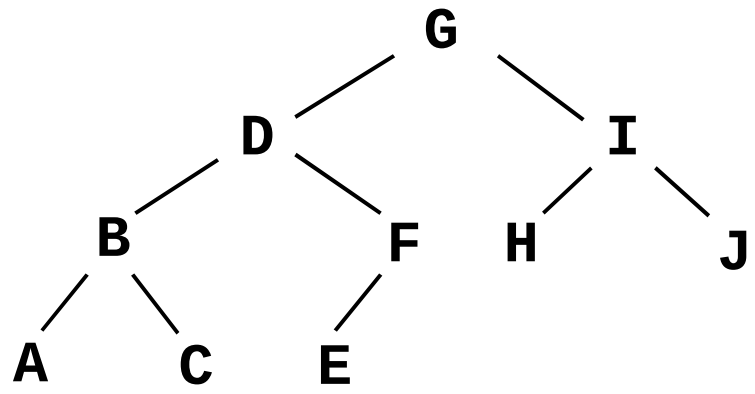
- In een binary tree heeft elke node 0, 1 of 2 'kinderen'
- Elke node heeft 0 of 1 'ouder'
- Sortering in binary tree:
  - Linkse subtree:
    - alle kleinere elementen
  - Rechtse subtree:
    - alle grotere elementen

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

zijn beide NULL voor leaf nodes

Voordeel t.o.v. list: elementen worden gemiddeld gevonden in tijd evenredig met de diepte (= logaritme van het totale aantal elementen).

Binary tree traversal: inorder

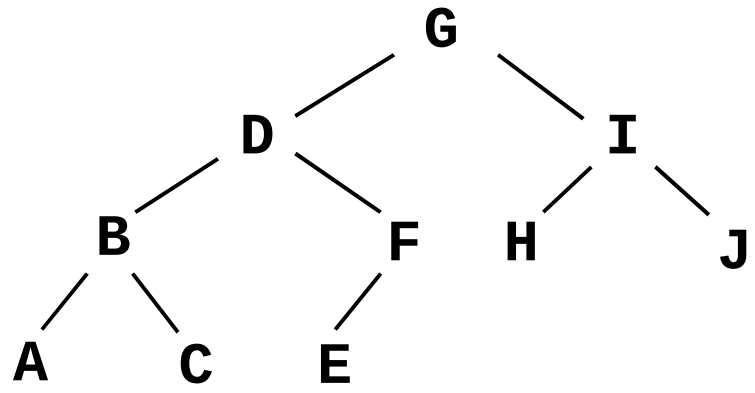



```
void inorder (struct node n)
   if (n != NULL) {
      inorder (n -> left);
      printf ("%c ", n -> data);
      inorder (n -> right);
   }
};
```

Output:

```
A B C D E F G H I J
```

Binary tree traversal: preorder

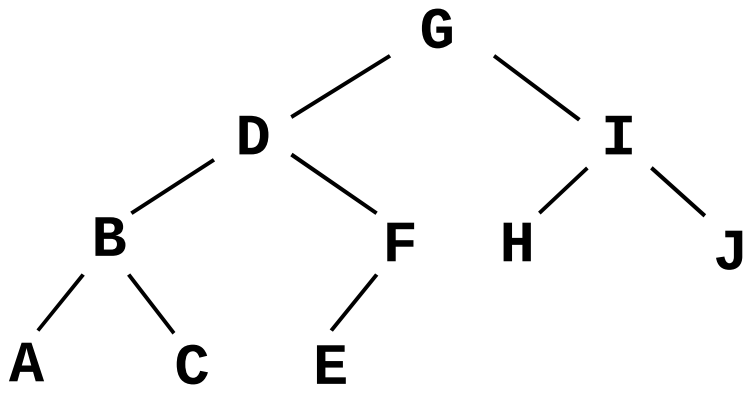



```
void preorder (struct node n)
   if (n != NULL) {
      printf ("%c ", n -> data);
      preorder (n -> left);
      preorder (n -> right);
   }
};
```

Output:

```
G D B A C F E I H J
```

Binary tree traversal: postorder



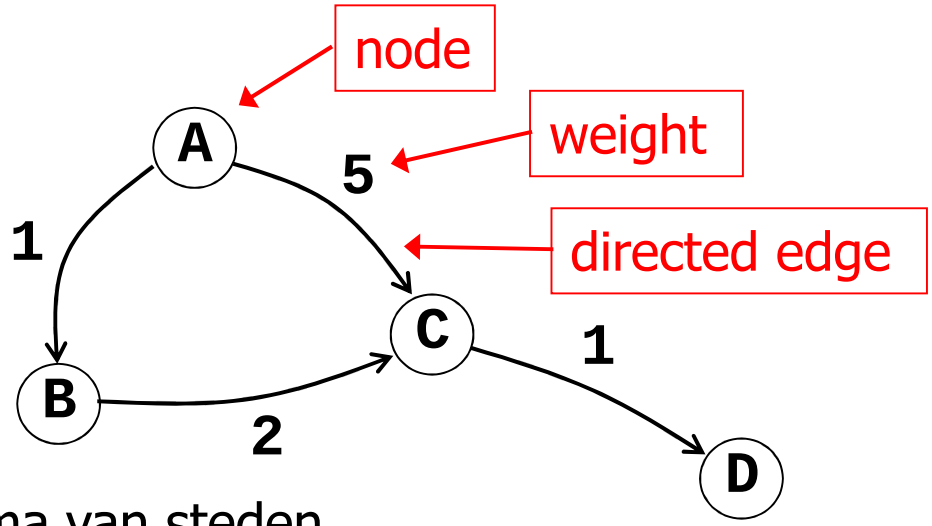

```
void postorder (struct node n)
   if (n != NULL) {
      postorder (n -> left);
      postorder (n -> right);
      printf ("%c ", n -> data);
   }
};
```

Output:

```
A C B E F D H J I G
```

Graphs

Graphs



Bijv.: routeplanner: schema van steden met onderlinge wegen (inclusief afstanden)

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

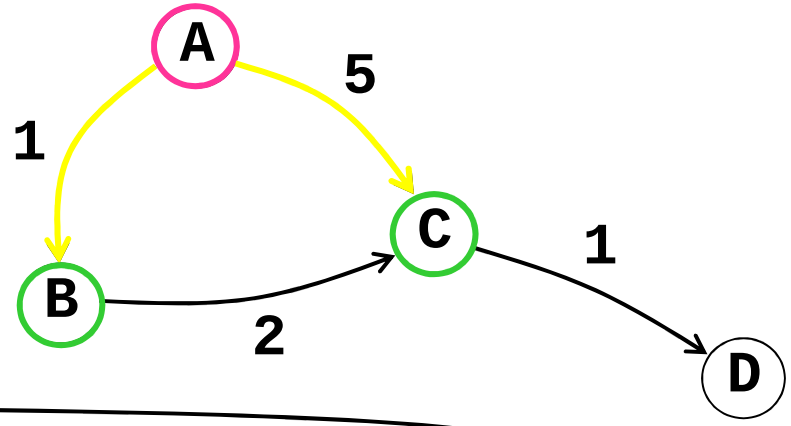
```
struct road {
    struct city * origin;
    struct city * destination;
    unsigned    length;
    struct road * next;
};
```

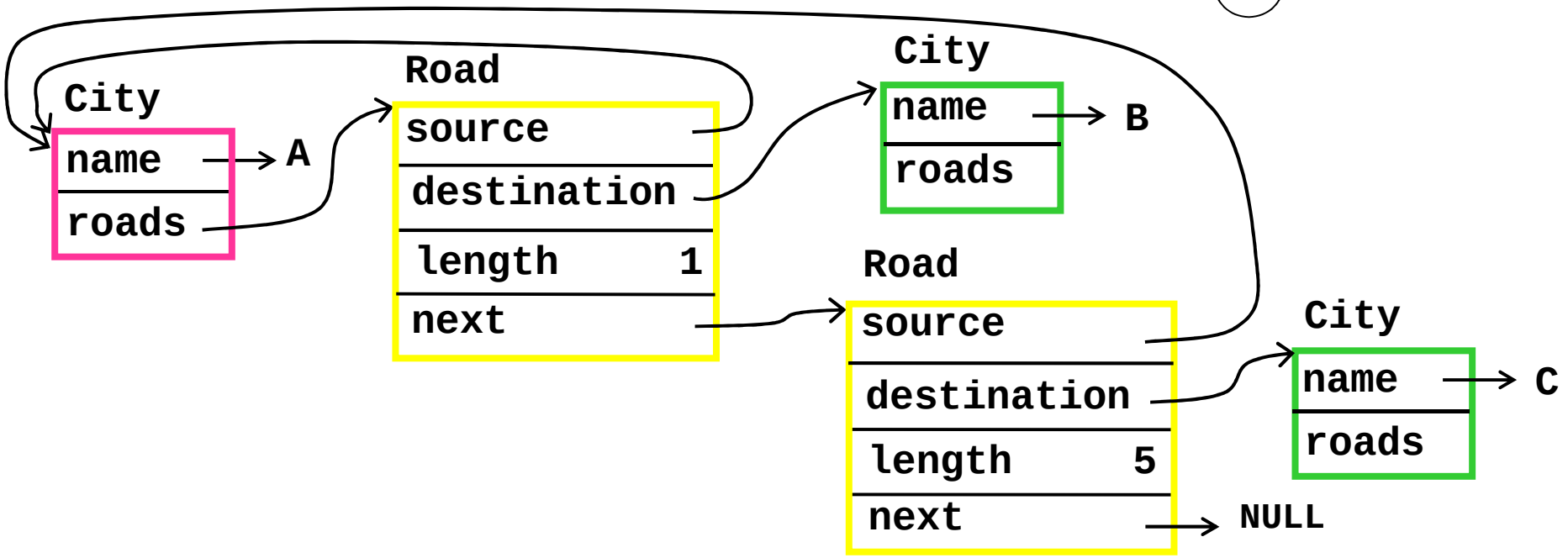
Dus hier:
- node → city
- edge → road
- weight → length

elke 'city' heeft een lijst (linked list) van 'roads' die uit de stad vertrekken.

Graphs – implementatie

Voorbeeld: Datastructuur waarmee verbindingen tussen A en buren wordt beschreven.





Input/Output en het Operating System

Input/output m.b.v. 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 */

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

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

FILE is een structure gedefiniëerd in stdio.h

wanneer file niet geopend kan worden wordt NULL geretouneerd

files sluiten

fscanf retouneert aantal toegekende argumenten

Stdin, stdout en stderr

Er bestaan 3 voorgedefiniëerde file pointers:

Written in C	Name	Remark
stdin	standard input file	connected to keyboard
stdout	standard output file	connected to screen
stderr	standard error file	connected to screen

Toepasselijker op vorige slide was:

```
fprintf (stderr, "Usage: %s inputfile outputfile\n", argv[0]);
```

Merk op:

```
fscanf (stdin, ....)     is gelijk aan   scanf (....)
fprintf (stdout, ....)   is gelijk aan   printf (....)
```

sprintf en sscanf

sprintf en sscanf zijn string versies van printf en scanf:

```
int sprintf (char *s, const char *format, ...);
int sscanf (const char *s, const char *format, ...);
```

Voorbeeld

```
char str1[] = "1 2 3 go";
char str2[100];
char tmp[100];

sscanf (str1, "%d%d%d%s", &a, &b, &c, tmp);
sprintf (str2, "%s %s %d %d %d\n", tmp, tmp, a, b, c);
printf ("%s", str2);
```

Output:

```
go go 1 2 3
```

Systeem commando's

Met de bibliotheek functie

```
int system (const char *command);
```

kunnen Windows/Linux programma's worden uitgevoerd vanuit een C programma.

```
#include <stdlib.h>

int main (void)
{
   ...
   system ("date");    /* execute command 'date' */
   ...
}
```

Samenvatting

- Samengestelde datatypen:
  - Structures
  - Unions
- Datastructuren:
  - Lists
  - Trees
  - Graphs
- Input/Output en het Operating System

Woensdag 19 dec. 13.45 – 14.30 uur voortgangstoets
Vrijdag 28 dec. 24.00 uur deadline practicum opdrachten
Vrijdag 1 feb. 2012, 9.00 – 12.00 uur tentamen