6 Gelinkte lijsten en bit manipulation

Oefening 36

```
/* De pointer l is uiteraard niet gewijzigd; enkel k.
   De pointer l is dus nog steeds de nullpointer.     */
```

Oefening 37

```
/* definitie knoop toevoegen */
void voeg_vooraan_toe(int x,knoop ** pl){
     knoop * even_bijhouden = *pl;
     *pl = (knoop*) malloc(sizeof(knoop));
     (*pl)->d = x;
     (*pl)->opv = even_bijhouden;
}

void print_lijst(const knoop * l){
     while( l != 0){
          printf("%d ",l->d);
          l = l->opv;
     }
     printf("X\n");
}

void print_lijst_recursief(const knoop * l){
     if( l!=0 ){
          printf("%d ",l->d);
          print_lijst_recursief(l->opv);
     }
     else{
          printf("X\n");  /* 'X' om einde van de lijst aan te geven,
                             zodat ook een lege lijst zichtbaar is */
     }
}

/* Niet correct:  void vernietig_lijst(knoop *l)
   Reden: l moet ook wijzigen in het hoofdprogramma (wordt 0)
   (en niet wijzen naar een vernietigde knoop)                          */

void vernietig_lijst(knoop **pl) {
     knoop *h;
     while (*pl != 0) {
          h = *pl;
          *pl = h->opv;
          printf("Ik geef knoop %d vrij\n",h->d); /*ter info*/
          free(h);
     }
}

/* aanvullingen in het hoofdprogramma:                                 */
     voeg_vooraan_toe(7,&l);  /* geef &l mee; aan een kopie van de pointer l
                               * wijzigingen aanbrengen helpt immers niet
                               * (zie vorige oefening!!)                */
     voeg_vooraan_toe(3,&l);
     print_lijst(l);          /* geef l mee; er moet toch niets veranderen
                               * aan dit adres                          */
     print_lijst_recursief(l);
```

```
/* toevoegen                                                              */
     vernietig_lijst(&l);
```

Oefening 38

```
/* werk verder op vorige oefening */

/* lijst met elementen in stijgende volgorde, dubbels mogelijk */
knoop* maak_gesorteerde_lijst_automatisch(int aantal, int bovengrens){
     knoop * l = 0;
     int getal = bovengrens;
     int i;
     for(i=0; i<aantal; i++){
          getal -= rand()%3;
          voeg_vooraan_toe(getal,&l);
     }
     return l;
}

/* Omdat je de eerste knoop nooit zal verwijderen,
   hoef je niet per se via knoop** door te geven      */
void verwijder_dubbels(knoop * l){
    while(l != 0){
          while (l->opv != 0 && l->opv->d == l->d){
               knoop * magweg = l->opv;
               l->opv = magweg->opv;
               free(magweg);
          }
          l = l->opv;
     }
}

/* HIERONDER EEN ALTERNATIEF, zie je wat er hier gebeurt?     */
void verwijder_dubbels_alternatieve_versie(knoop * l){
     /* h staat aan het begin van de gelijke waarden */
     /* m staat aan het einde van de gelijke waarden */
     knoop * h = l;
     knoop * m = l;
     while(m != 0){
          while(m != 0 && h -> d == m -> d){
               m = m -> opv;
          }
          if(h -> opv != m){
               knoop * magweg = h -> opv;
               knoop * einde_magweg = magweg;
               while(einde_magweg != 0 && einde_magweg->opv != m){
                    einde_magweg = einde_magweg -> opv;
               }
               einde_magweg -> opv = 0;
               vernietig_lijst(&magweg);
          }
          h -> opv = m;
          h = m;
     }
}

/* aanvulling in het hoofdprogramma:                        */
verwijder_dubbels(l);
/* toevoegen:                                                  */
vernietig_lijst(&l);
```

Oefening 39

```
/* werk voort op voorgaande oefening */
t/* bouw nog veiligheid in: geen twee dezelfde lijsten mergen! */
knoop * merge(knoop ** pa, knoop ** pb){
     knoop * l;
     knoop * i = *pa;
     knoop * j = *pb;
     knoop * k;
     if(*pa == 0){
          l = *pb;
          *pb = 0;
          return l;
     }
     else if(*pb == 0){
          l = *pa;
          *pa = 0;
          return l;
     }

     if((*pa)->d < (*pb)->d){
          l = *pa;
          i = (*pa)->opv;
          k = l;
     }
     else{
          l = *pb;
          j = (*pb)->opv;
          k = l;
     }
     while(i != 0 && j != 0){
          if(i->d < j->d){
               k -> opv = i;
               i = i -> opv;
          }
          else{
               k -> opv = j;
               j = j -> opv;
          }
          k = k -> opv;
     }
     if(i != 0){
          k -> opv = i;
     }
     if(j != 0){
          k -> opv = j;
     }

     *pa = 0;
     *pb = 0;
     return l;

}


/* vul het hoofdprogramma aan */
knoop * mn = merge_bis(&m,&n);

/* op het einde */
vernietig_lijst(&mn);
```

Oefening 40

```
void voeg_toe(int getal, knoop ** pl){
     knoop * naar_achter;
     while(*pl && (*pl)->d < getal){
          pl=&((*pl)->opv);
     }

     naar_achter = *pl;

     *pl = (knoop*) malloc(sizeof(knoop));
     (*pl)->d = getal;
     (*pl)->opv = naar_achter;
}

/* toevoegen in het hoofdprogramma:   */
     free_lijst(&l);
```

Oefening 41

```
void free_lijst(knoop ** pl){
     if(*pl != 0){
          free_lijst(&((*pl)->opv));
          printf("ik geef knoop met inhoud %d vrij\n",(*pl)->d);
          free(*pl);
          *pl = 0;
     }
}

/* toevoegen in het hoofdprogramma:   */
free_lijst(&l);
```

Oefening 42

```
void verwijder(int x, knoop **pl){
     knoop *te_verwijderen;
     while (*pl && (*pl)->d < x){
          pl = &((*pl)->opv);
     }
     if(*pl && (*pl)->d ==x ){
          te_verwijderen = *pl;
          (*pl)= (*pl)->opv;
          free(te_verwijderen);
     }
     else{
          printf("\n%i niet in lijst ",x); /* ENKEL IN TESTFASE LATEN STAAN
                                            * - of exceptie werpen
                                            * - of returntype van maken */
     }
}

/* toevoegen in het hoofdprogramma:   */
     verwijder(2,&l);  /* kies zelf wat je hier verwijdert - test grondig uit! */

     free_lijst(&l);
```

Oefening 43

```
/*  Oplossing wordt niet gepubliceerd.     */
```

Oefening 44

```
#include <stdio.h>
typedef unsigned int uint;
const int LENGTE = sizeof(uint)*8;

int bit_i(uint x, int i){
    /* alternatief:                              */
    /* return (x & (1<<i)) >> i;                 */
    return (x >> i) & 1;
}

void schrijf(uint x, int lengte){
    int i;
    int viervoud = lengte/4*4;
    for(i=lengte-1; i>=viervoud; i--){
        printf("%u",bit_i(x,i));
    }
    for(i=viervoud-1; i>=0; i-=4){
        printf(" ");
        int k;
        for(k=0; k<4; k++){
            printf("%u",bit_i(x,i-k));
        }
    }
    printf("    %u\n",x);
}
uint eenbit(int i){
    return 1<<i;
}

int aantal_eenbits(uint getal){
    int teller=0;
    while (getal) {
        teller+=(getal&1);
        getal>>=1;
    }
    return teller;
}

uint bit_i_aangezet(uint x, int i){
    return x | eenbit(i);
}

/* niet expliciet gevraagd; wel nuttig  */
uint alle_bits_omgedraaid(uint x){
    return ~x;
}

uint bit_i_uitgezet(uint x, int i){
    return x & ~(eenbit(i));
}

uint bit_i_gewisseld(uint x, int i){
    return x ^ eenbit(i);
}
```

```
int zijn_gelijk(uint a, uint b){
    return a^b;
}

int is_even(uint g){
        return 1 & ~(g&1);
}

/* Antwoord op de voorlaatste vraag:

n & (n-1) == 0 betekent dat n en n-1 nergens beide een bit gelijk hebben aan 1. (**)

Voorbeeld: stel
n   = abcde1000
n-1 = abcde0111

de bits die meer naar links staan (abcde) kunnen niet 1 zijn, want dat zou in
tegenspraak zijn met de eerste vaststelling (**).

Dus moeten ze allemaal 0 zijn, zodat n van de vorm 000001000 is.
Dus n heeft slechts 1 bit die aanstaat. Met andere woorden:
de test n & (n-1) == 0 gaat na of n een macht is van 2.                    */

int product(int a,int b){
        int res=0;
        while (a>0){
                if (a&1==1)
                        res+=b;
                b<<=1;
                a>>=1;
        }
        return res;
}

int main(){
    uint k = 49;
    int tel;
    schrijf(k,16);
    tel = aantal_eenbits(k);
    k = bit_i_uitgezet(k,5);
    schrijf(k,16);
    k = bit_i_aangezet(k,6);
    schrijf(k,16);
    schrijf(eenbit(6),16);
    schrijf(k^k,16);
    k = product(5,8);
    printf("%d=40",k);
    return 0;
}
```