Concepten C

Basisconcepten C

Conditionele uitdrukking: *voorwaarde* ? *uitdrukking 1* : *uitdrukking 2*

```
printf("Het teken van %d is: %c", a, a>0 ? '+' : (a<0 ? '-' : ' '));
```

```
i=0
while (i<10)
    printf("%d\n",i++);
```

Klassieke while loop:
Controle op voorwaarde en dan voeren we het vervolg van de code uit.

```
i=0
do {
    printf("%d\n",i++));
} while (i<10);
```

Do-while loop:
We voeren het vervolg van de code uit en controleren dan pas op de voorwaarde.

Legen van de buffer bij het inlezen van woorden/strings

```
while (getchar() != '\n')
      ;
```

Nooit de pow functie uit <math.h> gebruiken, eigen functie maken:

```
int my_pow(int x, int n) {
      if (n == 0)
            return 1;
      else
            return x*my_pow(x,n-1);
}
```

Pointers

```
//gewone pointer
int g = 5;
int *p;
p = &g;
//nullpointer
double *p;
p = 0; of p = NULL;
//void-pointer
void *v;
v = &g;
//derefereren void pointer
printf("%d",*(int *)v);
```

Functie schrijven die twee waarden omwisselt dmv pointers (call by reference)

```
void wissel(int *a, int *b) {
      int hulp = *a;
      *a = *b;
      *b = hulp;
}
int main() {
      int g1 = 1, g2 = 2;
      wissel(&g1, &g2);
      return 0;
}
```

Pointer naar const

```
int n = 4; const int *q = &n;
//door const toe te voegen kunnen we niks wijzigen
(*q++); // leidt tot error, const kan niet wijzigen!
int *p = &n; (*p)++; *p = *q;
*q = *p; // leidt tot error, const kan niet wijzigen!
```

Pointer verplaatsen: pointer van een pointer meegeven!

Constante pointer: pointer kan niet aangepast worden, waarde wel

Functie-pointer

```
return-type (*fname)([prototype-lijst]);
```

C-string

Pointer naar karakter van karaktersliert, gesloten met nullpointer!

```
char s[80];
//inlezen scanf
scanf("%s",s);
scanf("%79s",s); //veiliger door beperking
//inlezen (f)gets
gets(s); //kan buiten bereik gaan!
fgets(s,80,stdin); //veiliger, kan niet buiten bereik gaan
```

argc en argv

```
int main(int argc, char **argv) {
      return 0;
}
// int argc = hoeveelheid ingegeven strings + bestandsnaam
// char **argv = pointer naar de eerste ingegeven strings
in een array van chars
```

Structs

```
struct naam {
      type_1 naam_1;
      type_2 naam_2;
};
typedef struct naam naam;
```

Versie 1 van de definitie van een struct

```
typedef struct {
      type_1 naam_1;
      type_2 naam_2;
} naam;
```

Versie 2 van de definitie van een struct, gebruikelijke notatie in C

Toegang krijgen: .operator

Aan elkaar gelijk stellen: = operator

Vergelijken: variabelen afzonderlijk vergelijken

Initialisatie bij declaratie:

```
typedef struct {
      double x,y;
} punt;
punt vb = {1.1,2.2};
```

Dynamisch geheugenbeheer

Toewijzen en vrijgeven geheugen

```
punt p = lees_punt();
punt lees_punt() {
      punt *p = malloc(sizeof(punt));
      //geheugen reserveren, anders geven we locale variabele terug
      scanf("%lf",&p->x); scanf("%lf",&p->y);
      return p;
free(p); //geheugen MOET achteraf vrijgegeven worden
```

Hergebruiken geheugen

```
char *p = calloc(11,sizeof(char));
strcpy(p,"1234567890");
char *nieuw = realloc(p,20);
if (nieuw) {
      strcat(nieuw,"langer");
      puts(nieuw);
      free(nieuw);
} else {
      free(p);
}
```

Gelinkte lijsten

Gelinkte lijst bestaat uit knopen. Elke knoop bevat data en een pointer of meerdere pointers naar een buurknoop of buurknopen.
Enkelgelinkte lijst: elke knoop bevat een pointer naar zijn opvolger.
Dubbelgelinkte lijst: elke knoop bevat twee pointers: naar zijn voorganger en zijn opvolger.

Definitie

```
typedef struct knoop knoop;
struct knoop {
      int getal;
      knoop *volgend;
};
```

Lijst overlopen

```
while (lst) {
      printf("%d\n", lst->getal);
      lst = lst->volgend;
}
```

Achteraan toevoegen

```
knoop* maak_lijst() {
      knoop *l = 0;
      int g; scanf("%d",&g);
      if (g) {
            l = malloc(sizeof(knoop));
            knoop *h = l; h->getal = g;
            scanf("%d",&g);
            while (g) {
                  h->volgend = malloc(sizeof(knoop));
                  h = h->volgend; h->getal = g; scanf("%d",&g);
            }
            h->volgend = 0;
      }
      return l;
}
```

Vooraan toevoegen

```
void voeg_vooraan_toe(knoop **pl, const char *s)
{
    knoop *even_bijhouden = *pl;
    *pl = malloc(sizeof(knoop));
    strcpy((*pl)->woord, s);
    (*pl)->volgend = even_bijhouden;
}
```

Lijst vernietigen

```
void vernietig_lijst(knoop **pl) {
      while (*pl) {
            knoop *h = *pl;
            *pl = h->next;
            free(h);
      }
}
```

Bit manipulation

Shift left

```
int a = 3;
for (int i=1 ; i<3 ; i++)
a <<= i;
```

Shift right

```
int a = 192;
for (int i=1 ; i<3 ; i++)
a >>= i;
```