Herhaling

Functies

Gestructureerd programmeren **bedoelt dat een groot programma in diverse *sub-programma's* opgedeeld wordt. Een eenheid van instructies kan in een functie samengevat worden.**

<type> <functie naam> (<type> <parameter naam>)

voorbeeld:

```
int power(int base, int n)
 {
   int i, p;
   p = 1;
   for ( i=1; i <= n; i++ )  p = p * base;
   return(p);
 }
```

Deze functie berekent de <*n*>de macht van het getal <*base*>.
Met return wordt een waarde aan het uitvoerende programma terug gegeven.

```
int main()
 {
  int m;
  m = power(2,3)   /* berekent 2^3 en geeft het resultaat naar m terug */
 }
```

Lokale en globale variabelen

De definitie van een variabel geldt in een omgeving die door de haakjes { en } aangegeven wordt. Dit is een lokale variabele.

Een variabele die buiten alle {} gedefinieerd is geldt in het hele programma. Dit is een globale variabele.

voorbeeld:

```
int global;   /* een globale variabele */

int een_functie(int i)
 {
  int local;  /* een lokale variabele */
 }

main()
 {
  int local;
 /* nog een lokale variabele. Dit heft niets met „local“ in een_functie te doen */
 }
```

Data types

Namen van variabelen

A..Z, a..z, 0..9, _ een variabel begint met A..Z of a..z
31 tekens zijn significant.

Traditioneel gebruiken wij
a..z voor variabelen en
A..Z voor constanten.

Worden zoals *if*, *while*, *for* kunnen niet voor variabelen gebruikt worden.

Alle variabelen moeten voor gebruik gedefinieerd worden.

Data types

De basistypen zijn:
char
int
float
double

Variaties van deze basistypen zijn: short en long
long int, short int, long double

Bovendien hebben wij nog signed en unsigned voor elke char en int type.

voorbeeld:
char: -128 .. 0 .. 127 en
unsigned char 0 .. 255

precision.c

```
#include<stdio.h>

int main()
 {
  int i;
  float a;
  double b;
  long double c;

  i=            5/3;
  a=(float)     5/3;
  b=(double)    5/3;
  c=(long double)5/3;

  printf("%i %30.28e %30.28e %30.28Le \n",i,a,b,c);
 }
```

```
> gcc -o precision precision.c
> ./precision
1 1.66666669302368164062500000e+00 1.66666666666666674068153497 50e+00
1.66666666666666666305265943e+00
>
```

Data types

Om tekst op te slaan worden char arrays gebruikt.

char text[80];

In deze variabel zijn 80 tekens opgeslagen.

Een string eindigt in het geheugen met een \0.

tt[] = “een tekst“;

string: een veld van tekens of een vector van teekens.

Constanten

Constanten worden met #define gedefinieerd.

voorbeeld:

```
#define LENGTH 80  /* een integer waarde */
char text[LENGTH];

#define PI 3.1415     /* een floating point waarde */

#define ABC 0x00ff  /* een hexadecimaal cijfer */

#define STRING “een tekst“  /* een string constante */
```

Ook een variabele kan als een constante gedefinieerd worden

```
const double e = 2.71828182845905;
```

Constanten

Let op: “x“ en ‘x‘ zijn niet het zelfde!

#define AA “x“ is in het geheugen: x\0
(de ““ worden niet opgeslagen)

#define AA ‘x‘ is in het geheugen een integer waarde

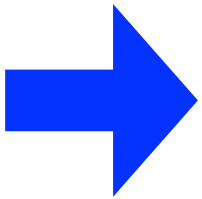
taak3.c

```
#include <stdio.h>

#define ABC 'x'
#define CDE "x"

int main()
{
 char cc = ABC;
 char text[] = CDE;

 printf("%i \n",cc);
 printf("%c \n",cc);
 printf("%i %i \n", text[0], text[1] );
 printf("%s \n", text);
}
```



```
120
x
120 0
x
```

Operatoren

Rekenkundige operatoren

Rekenkundige operatoren zijn +, -, *, / en %
% geeft de rest van een deling. `a=5%3;` **--> a=2**

prioriteit: + - **unitaire operatoren (voorteken)**
* / % **binaire operatoren**
+ - **binaire operatoren**

Operatoren

Operatoren voor vergelijkingen

> >= < <= hogere prioriteit

== != lagere prioriteit

Rekenkundige operatoren hebben een hogere prioriteit dan vergelijkingen.

a < b-1 --> a < (b-1)

Let op!

a = b kopieert de waarde van b naar a, dit is een toewijzing

a==b vraagt: is a het zelfde als b?

Dit is een vergelijking.

Operatoren

Logische operatoren: **&& en** **|| of**

Uitdrukkingen die met && of || gekoppeld zijn worden steeds van links naar rechts geëvalueerd.

&& heft een hogere prioriteit dan ||. Allebij hebben een lagere prioriteit dan vergelijkingsoperatoren.

voorbeeld: **if (a>10 && a<15) heft geen extra haakjes nodig.**

! **negatie operator, een unitair operator**
!x **geeft 0 wanneer x != 0 en 1 anders**

voorbeeld: **if (!valid)** **in plaats van if (valid == 0)**

voorbeeld:

```
#define OK 0
if ( !OK )
```

Type conversie

(type name) expression

taak4.c

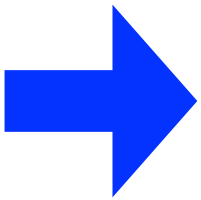
```
#include <stdio.h>

int main()
{
 int i,k;
 float a,b;

 i=2;
 a=10.5;

 b=a+i;                  /* hier wordt de type impliciet geconverteerd */
 printf("%f \n",b);

 k =  (int) a % i;       /* % werkt alleen voor char en int variabelen */
 printf("%i \n",k);
}
```



```
12.500000
0
```

Increment en decrement operator

De operatoren ++ en -- kunnen voor of naar een variabele staan.

```
n=5;
x = ++n;
```

--> x =6 en n=6.

n wordt teerst verhoogt en daarnaar naar x toegewezen

```
n=5;
x = n++;
```

--> x=5 en n=6.

n wordt teerst naar x toegewezen en daarnaar om 1 verhoogt

Toewijzingen

```
a = b + c;
```

```
i = i +2;
```
of korter
```
i += 2;
```

Deze toewijzingsoperator bestaat voor de meeste binaire operatoren, zoals + - * / %

Let op: `x *= y+1`
bedoelt `x = x * (y+1)`
en niet `x = x * y + 1`

Het resultaat van een toewijzing kan ook in een vergelijking gebruikt worden.

voorbeeld:

```
while ( ( c = getchar() ) != EOF)
```

Leest een teken van de standaard input tot het einde van het bestand.

Voorwaardelijke toewijzingen

```
if (a > b) z=a;
   else    z=b;
```

berekent het maximum van a en b. Of korter

```
z = ( a>b ? a : b);
```

(*expr1* ? *expr2* : *expr3*)

Teerst wordt *expr1* berekend. Als dit niet 0 is wordt *expr2* berekend, anders *expr3*.

Voorwaardelijke toewijzingen

Met een voorwaardelijke toewijzing kunt u heel compact code schrijven

```
for ( i=0; i<N; i++)
  printf("%6i%c", a[i], (i%10==9 || i==N-1 ) ? '\n' : ' ' );
```

Dit geeft een vector van integer waarden uit.
10 waarde per rij, naar elke rij wordt een \n uitgegeven.

Prioriteit van operatoren

1. Function calls, array subscription () [] -> .
2. Unary operators (right to left) ! ~ ++ -- + - & (type) sizeof
3. Multiplication, division, and modular division expressions * / %
4. Addition and subtraction expressions + -
5. Bitwise shifting expressions << >>
6. Greater-than, less-than, greater-than-or-equal-to, and less-than-or-equal-to expressions < <= > >=
7. Equal-to and not-equal-to expressions == !=
8. Bitwise AND expressions &
9. Bitwise exclusive OR expressions ^
10. Bitwise inclusive OR expressions |
11. Logical AND expressions &&
12. Logical OR expressions ||
13. Conditional expressions (using ?:). When used as subexpressions, these are evaluated right to left ? :
14. All assignment expressions.When multiple assignment statements appear as subexpressions in a single larger expression, they are evaluated right to left.
    = += -= *= /= %= &= ^= |= <<= >>=
15. Comma operator expressions ,

Hoofdstuk 3: Controlestructuren

Instructies en blokken

Naar elke instructie staat een ; puntkomma

Verschillende instructies worden door de accolades {}
in een block samengevat.
Een blok is het zelfde als een enkele instructie.

Naar de } aan het einde van een blok staat geen ;

if else

```
if ( expression)
   statement1;
 else
   statement2;
```

Attentie: `if (expression)` **is het zelfde als if** `(expression != 0 )`

```
if (n>0)
   if (a>b) z=a;
   else z=b;
```

de else hoort bij de laatste if

als u dit wilt

```
if (n>0)
  {
   if (a>b) z=a;
  }
 else z=b;
```

hebben wij accolades nodig

else if

Een beslissing met verschillende alternatieven.

```
if ( expression1)
   statement;

else if (expression2)
   statement;

else if (expression3)
   statement;

else
   statement;
```

```
/* binary search */
/* search for an element in a sorted list */

#include <stdio.h>
#define MAX 10

int binsearch(int x, int v[], int n)
 {
  int low=0, high=n-1, mid;

  while( low <= high)
   {
    mid=(low+high)/2;
    if( x < v[mid] )
     {
      high = mid -1;
     }
    else if ( x > v[mid] )
     {
      low = mid +1;
     }
    else   /* element found */
     {
      return(mid);
     }
   }
  return(-1); /* not found */
 }
```

```
int main()
 {
  int x=27;
  int a[]={1,3,6,22,27,30,31,32,33,50};

  printf(" %i is at position %i in list\n",
            x,binsearch(x,a,MAX) );
 }
```

example16.c

switch

De switch instructie wordt voor een selectie van verschillende constante integer waarden gebruikt.

```
switch ( expression)
 {
  case const-expr : statement;
    break;
  case const-expr : statement;
    break;
  default: statement;
 }
```

switch

voorbeeld:

```
char c;

switch (c)
 {
  case ‘a‘: doe_iets;
    break;
  case ‘b‘: doe_iets_anders;
    break;
  default: do_niets;
 }
```

Let op, break is belangrijk hier.

```
switch (c)
 {
   case ‘0‘: case ‘1‘: case ‘2‘:
     do_iets;
     break;
   default: doe_iets_anders;
 }
```

Voor 0,1 en 2 wordt doe_iets uitgevoerd.

```
/* count the number of digits, white space and other characters */

#include <stdio.h>

int main()
 {
  int c, i, nwhite=0, nother=0, ndigit[10]={0,0,0,0,0,0,0,0,0,0};

  while( ( c=getchar()) != EOF )
   {
    switch(c)
     {
      case '0': case '1': case '2': case '3': case '4':
      case '5': case '6': case '7': case '8': case '9':
        ndigit[c-'0']++;
       break;
      case ' ': case '\n': case '\t':
        nwhite++;
       break;
      default:
        nother++;
       break;
     }
   }

  printf("digits=");
  for(i=0; i<10; i++)
    printf(" %d", ndigit[i]);
  printf(", white space = %d, other = %d\n", nwhite, nother);
}
```

example17.c

switch statement

```
int main()
 {
   char c='a';
   int i;

   switch(c)
    {
      case 'a': i=5;
      case 'b': i+=1;
        break;
      default: i=0;
    }
}
```

hier staat geen break!

Wat is de waarde van i?

A) i=0
B) i=5
C) i=6

loops (lussen) while en for

```
while (expression)
  statement;
```

Herhaalt statement zolang expression niet 0 is.

```
for ( expr1 ; expr2 ; expr3 )
 statement;
```

expr1 t/m 3 kunnen leeg zijn.

Als expr2 leeg is wordt expr2 als waar aangenomen.
voorbeeld: `for( ; ; ) {}` is dus een eindeloze lus.

Verschillende instructies woorden door en , komma gescheiden.
voorbeeld:

```
for ( a=1, b=3 ; a<10 ;  a++, b--)
```

```
/* shellsort: sort a vector in increasing order */

#include<stdio.h>

void shellsort(int v[], int n)
 {
  int gap,i,j,temp;

  for(gap=n/2; gap>0; gap/=2)
   {
    for(i=gap; i<n; i++)
     {
      for(j=i-gap; j>=0 && v[j]>v[j+gap]; j-=gap)
       {
        temp=v[j];
        v[j]=v[j+gap];
        v[j+gap]=temp;
       }
     }
   }
  return;
 }
```

```
int main()
 {
  int i;
  int a[10]={5,27,3,81,9,17,102,1,33,15};

  shellsort(a,10);

  printf("sorted=");
  for(i=0; i<10; i++)
   {
    printf("%4d",a[i]);
   }
  printf("\n");
 }
```

example19.c

```
/* reverse the order of a string */

#include<stdio.h>
#include<string.h>

void reverse(char s[])
 {
  int c,i,j;

  for(i=0, j=strlen(s)-1; i<j; i++, j--)
   {
    c=s[i];
    s[i]=s[j];
    s[j]=c;
   }
  return;
 }
```

```
int main()
 {
  char test[]="This is a test";

  printf("before=%s\n",test);
  reverse(test);
  printf("after =%s\n",test);
 }
```

example20.c

loops (lussen) do while

```
do
   statement;
while ( expression );
```

Bij while en for wordt expression teerst getest,
daarnaar wordt de lus uitgevoerd.

Bij do{}while wordt teerst statement uitgevoerd en
daarnaar expression getest.
Als expression waar is wordt de lus herhaalt.

```
/* convert a number into a string */

#include<stdio.h>
#include<string.h>

void reverse(char s[])   /* reverse string */
 {
  int c,i,j;

  for(i=0, j=strlen(s)-1; i<j; i++, j--)
   {
    c=s[i];
    s[i]=s[j];
    s[j]=c;
   }
  return;
 }
```

```
int main()
 {
  int i;
  char text[50];

  i=20;
  int2string(i,text);
  printf("%i  %s \n",i,text);
  i=-45;
  int2string(i,text);
  printf("%i  %s \n",i,text);
 }
```

```
/* convert number to string */
void int2string(int n, char s[])
 {
  int i=0,sign;

  if((sign=n) <0)   /* remember sign */
   {
    n=-n;
   }

  do   /* generate digits from right to left */
   {
    s[i++] = n%10+'0';
   } while(( n/=10) >0);

  if (sign<0)
   {
    s[i++]='-';
   }
  s[i]='\0';
  reverse(s);
 }
```

example21.c

break en continue

Met break wordt een loop (for, while en do) vroegtijdig beëindigt.

```
for ( a=1; a<10; a++)
   {
    doe_iets;
    if (error) break;
    doe_nog_iets;
   }
```

break kan ook binnen switch gebruikt worden.

continue in een for, while of do loop herhaalt de loop vroegtijdig. De volgende herhaling wordt onmiddellijk begonnen.

```
for ( a=1; a<10; a++)
   {
    doe_iets;
    if (a==3) continue;
    doe_nog_iets_maar_niet_voor_3;
   }
```

```
/* remove white space, tabulator and new line from the end of a string */
#include<stdio.h>
#include<string.h>

int trim(char s[])
 {
  int n;

  for(n=strlen(s)-1; n>=0; n--)
   {
    if(s[n]!=' ' && s[n]!='\t' && s[n]!='\n' )
     {
      break;
     }
   }
  s[n+1]='\0';
  return(n);
}

int main()
 {
  char text[]="This is a test     \n";

  printf("before >%s<\n",text);
  trim(text);
  printf("after  >%s<\n",text);
 }
```

example22.c