Herhaling

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;
```

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;
 }
```

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--)
```

loops (lussen) do while

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

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

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

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;
   }
```

Functies

In C worden verschillende taken in een programma in subroutines (functies) opgesplitst.

```
<type> <function name> (<type> parameter, <type> parameter )
  {
   return(expression);   /* heft de type van <function name> */
  }
```

Functies met de type void hebben geen return waarde.

Algemeen structuur van een programma

```
/* library headers */
#include <stdio.h>
#include ...

/* define constants */
#define OK 1
#define ...

/* define global variables */
int global;
float ...

/* define functions */
int function1(int i)
 {
  return(OK);
 }

float function2(float x)
 {
  return()
 }

/* main program */
int main()
 {
  int error_code=0;
  return(error_code);
 }
```

Functies

```
/* library headers */
#include <stdio.h>

/* define constants */
#define OK 1

/* define global variables */

/* define functions */
float square(float x)
 {
  return(x*x);
 }

/* main program */
int main()
 {
  float a,b;
  a=2;
  b=square(a);
  printf(" %f ^2 = %f \n",a,b);
 }
```

x=2

2*2 = 4

a=2

b = square(2) = 4

a=2 b=4

Header files (definitie bestanden)

Standaard functies zijn in bibliotheken (libraries) samengevat.

header files worden met #include <filename> in een programma ingevoegd.

voorbeld:

```
#include<stdio.h>
#include<math.h>
```

C standard library

Name	From	Description
<assert.h>		Contains the assert macro, used to assist with detecting logical errors and other types of bug in debugging versions of a program.
<complex.h>	C99	A set of functions for manipulating complex numbers.
<ctype.h>		Defines set of functions used to classify characters by their types or to convert between upper and lower case in a way that is independent of the used character set (typically ASCII or one of its extensions, although implementations utilizing EBCDIC are also known).
<errno.h>		For testing error codes reported by library functions.
<fenv.h>	C99	Defines a set of functions for controlling floating-point environment.
<float.h>		Defines macro constants specifying the implementation-specific properties of the floating-point library.
<inttypes.h>	C99	Defines exact width integer types.
<iso646.h>	NA1	Defines several macros that implement alternative ways to express several standard tokens. For programming in ISO 646 variant character sets.
<limits.h>		Defines macro constants specifying the implementation-specific properties of the integer types.
<locale.h>		Defines localization functions.
<math.h>		Defines common mathematical functions.
<setjmp.h>		Declares the macros `setjmp` and `longjmp`, which are used for non-local exits.
<signal.h>		Defines signal handling functions.
<stdalign.h>	C11	For querying and specifying the alignment of objects.
<stdarg.h>		For accessing a varying number of arguments passed to functions.
<stdatomic.h>	C11	For atomic operations on data shared between threads.
<stdbool.h>	C99	Defines a boolean data type.
<stddef.h>		Defines several useful types and macros.
<stdint.h>	C99	Defines exact width integer types.
<stdio.h>		Defines core input and output functions
<stdlib.h>		Defines numeric conversion functions, pseudo-random numbers generation functions, memory allocation, process control functions
<stdnoreturn.h>	C11	For specifying non-returning functions.
<string.h>		Defines string handling functions.
<tgmath.h>	C99	Defines type-generic mathematical functions.
<threads.h>	C11	Defines functions for managing multiple Threads as well as mutexes and condition variables.
<time.h>		Defines date and time handling functions
<uchar.h>	C11	Types and functions for manipulating Unicode characters.
<wchar.h>	NA1	Defines wide string handling functions.
<wctype.h>	NA1	Defines set of functions used to classify wide characters by their types or to convert between upper and lower case

from Wikipedia

C standard library

de belangrijkste header files

stdio.h	input & output	printf, scanf
ctype.h	tests voor character classes (tekens)	islower
string.h	strings (tekens)	strlen, strcpy
math.h	wiskundige functies	sin, pow
stdlib.h	hulp functies	rand, atoi
signal.h	signal handling	
time.h	functies voor tijd	time
limits.h	grenzen van variabelen typen	INT_MAX, INT_MIN
float.h	grenzen van variabelen typen	FLT_MAX, FLT_MIN

math.h

	Function	Description
	abs labs llabs	computes absolute value of an integer value
	fabs	computes absolute value of a floating point value
	div ldiv lldiv	computes the quotient and remainder of integer division
	fmod	remainder of the floating point division operation
	remainder	signed remainder of the division operation
	remquo	signed remainder as well as the three last bits of the division operation
	fma	fused multiply-add operation
	fmax	larger of two floating point values
	fmin	smaller of two floating point values
	fdim	positive difference of two floating point values
	nan nanf nanl	returns a not-a-number (NaN)
Exponential functions	exp	returns e raised to the given power
	exp2	returns 2 raised to the given power
	expm1	returns e raised to the given power, minus one
	log	computes natural (base e) logarithm (to base e)
	log10	computes common (base 10) logarithm
	log1p	computes natural logarithm (to base e) of 1 plus the given number
	ilogb	extracts exponent of the number
	logb	extracts exponent of the number
Power functions	sqrt	computes square root
	cbrt	computes cubic root
	hypot	computes square root of the sum of the squares of two given numbers
	pow	raises a number to the given power

from Wikipedia

math.h

Trigonometric functions	`sin`	computes sine
	`cos`	computes cosine
	`tan`	computes tangent
	`asin`	computes arc sine
	`acos`	computes arc cosine
	`atan`	computes arc tangent
	`atan2`	computes arc tangent, using signs to determine quadrants
Hyperbolic functions	`sinh`	computes hyperbolic sine
	`cosh`	computes hyperbolic cosine
	`tanh`	computes hyperbolic tangent
	`asinh`	computes hyperbolic arc sine
	`acosh`	computes hyperbolic arc cosine
	`atanh`	computes hyperbolic arc tangent
Error and gamma functions	`erf`	computes error function
	`erfc`	computes complementary error function
	`lgamma`	computes natural logarithm of the gamma function
	`tgamma`	computes gamma function
Nearest integer floating point operations	`ceil`	returns the nearest integer not less than the given value
	`floor`	returns the nearest integer not greater than the given value
	`trunc`	returns the nearest integer not greater in magnitude than the given value

	`round`
	`lround`
	`llround`
		returns the nearest integer, rounding away from zero in halfway cases

	`nearbyint`	returns the nearest integer using current rounding mode

	`rint`
	`lrint`
	`llrint`
		returns the nearest integer using current rounding mode with exception if the result differs

from Wikipedia

math.h

Floating point manipulation functions	frexp	decomposes a number into significand and a power of 2
	ldexp	multiplies a number by 2 raised to a power
	modf	decomposes a number into integer and fractional parts
	scalbn scalbln	multiplies a number by FLT_RADIX raised to a power
	nextafter nexttoward	returns next representable floating point value towards the given value
	copysign	copies the sign of a floating point value
Classification	fpclassify	categorizes the given floating point value
	isfinite	checks if the given number has finite value
	isinf	checks if the given number is infinite
	isnan	checks if the given number is NaN
	isnormal	checks if the given number is normal
	signbit	checks if the given number is negative

om de math library te gebruiken, hebben wij het linker commando -lm nodig

gcc -o naam naam.c -lm

voor meer informatie over libraries, zie ook cursus web pagina

from Wikipedia

Functies - Recursion

Een functie kan ook zichzelf oproepen.

voorbeeld: n! faculteit

```
double factorial( int n)
 {
  return( n==1 ? 1 : n*factorial(n-1) );
 }
```

Waarom is deze functie van de type double?

factorial.c

Nog een keer #define

Met #define kunnen constanten gedefinieerd woorden. Maar #define kan ook gebruikt worden om tekst te vervangen.

```
#define max(A,B) ( (A) > (B) ? (A) : (B) )
```

Dit is geen functie. De compiler vervangt max(A,B) met (...)

```
x=max(p+q, r+s);
```

wordt

```
x=((p+q) > (r+s) ? (p+q) : (r+s));
```

Let op:

```
max ( i++, j++)   /* verkeerd */
```

De grotere waarde wordt 2 keer verhoogd!

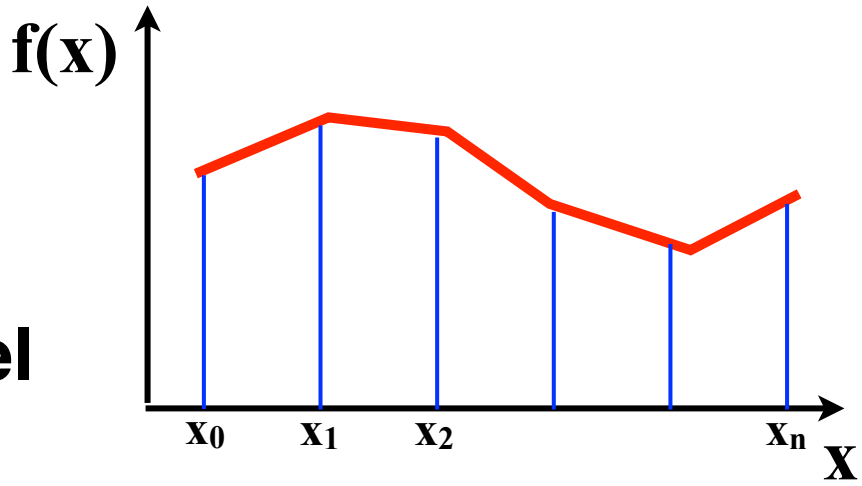
Numerieke algoritmen

(Voorbereiding van het computer practicum)

Integrale en de trapezium regel

De integraal

$$I = \int_a^b f(x)dx$$



kan met de trapezium regel berekend worden:

$$I = I_n = \frac{h}{2}\left(f(x_0) + 2f(x_1) + \ldots 2f(x_{n-1}) + f(x_n)\right)$$

voor $i = 1 \ldots n$ **met** $h = \frac{b-a}{n}$

$x_0 = a$ **en** $x_i = a + ih.$

Priemgetallen

Een algoritme om priemgetallen te berekenen:

herhaal n van 2 t/m max

herhaal i van n-1 t/m 1 (i--)

stop herhaling als n%i==0 % berekent de rest van een divisie

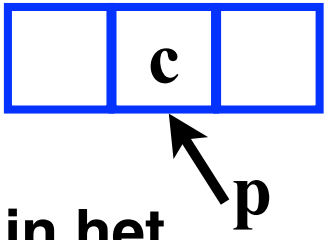
als i == 1 dan is n een priemgetal

Hoofdstuk 5:
Pointers en vectoren

Dit is het hart van de taal C. Als jullie dit begrijpen spelen jullie in de champions league.

Pointers en adressen

Een eenvoudige afbeelding van het geheugen



De adres-operator geeft het adres van een object in het geheugen.

p = &c;

Het adres van de variabele c wordt aan p toegewezen.

De inhoud-operator geeft de inhoud van een pointer.

a = *p;

voorbeeld:

```
int x=1, y=2;   /* variabelen x en y */
int *ip;        /* pointer naar int */

ip = &x;        /* ip wijst nu op x */
y = *ip;        /* wijst de inhoud van ip naar y toe */
*ip = 0;        /* x is nu 0 */
```

Pointers en adressen

int *ip is een nieuwe type van variabele, een pointer.

```
int *ip, x;
ip = &x;
```

als ip naar x wijst dan kan *ip in plaats van x gebruikt worden

```
*ip = *ip + 10;      x = x + 10;
*ip += 1;            x += 1;
```

pointers zijn gewone variabelen

```
int *ip, *iq;    /* 2 pointers */
ip = iq;         /* ip en iq wijzen nu naar het zelfde object */
```

Pointers en argumenten van functies

In C worden de waarde van argumenten aan een functie toegewezen.

voorbeeld:
Met de functie swap(a,b) willen wij de inhoud van a en b ruilen.

```
void swap(int x, int y)   /* dit werkt niet! */
 {
  int temp;
  temp = x;
  x = y;
  y = temp;    /* dit gebuurt alleen hier lokaal */
 }
```

Pointers en argumenten van functies

Om de inhoud van a en b te ruilen hebben wij pointers nodig.

```
void swap(int *px, int *py)    /* *px en *py ruilen */
  {
   int temp;

   temp = *px;
   *px = *py;
   *py = temp;
  }
```

Deze functie wordt met `swap(&a, &b)` opgeroepen. Dit geeft de adressen (of pointers) van `a` en `b` aan de functie. De parameters van deze functie zijn pointers.

Met pointer argumenten kan een functie waarde binnen de roepende functie veranderen.
Gewone argumenten zijn lokale kopieën van de waarde van de roepende functie.

Pointers en vectoren

In C bestaat een nauwe relatie tussen pointers en vectoren.
Elke operatie met vector indices kan ook met pointers geformuleerd worden.

`int a[10];` definieert een vector met 10 elementen.

```
a: |___|___|___|___|___|___|___|___|___|___|
   a[0] a[1] a[2] a[3] a[4] a[5] a[6] a[7] a[8] a[9]
```

```
int *pa;
pa = &a[0];
```
`pa` wijst nu naar het element `0` van `a`.

De toewijzing `x = *pa` kopieert de inhoud van `a[0]` naar `x`.

`pa+1` wijst naar het volgende element,
`pa+i` naar het *i*-de element.
Het adres van `a[i]` is `pa+i` en `*(pa+i)` geeft de inhoud van `a[i]`.

Pointers en vectoren

```
int a[10];
int *pa;
```

De naam van de variabele `a` geeft het adres van `a[0]`.

`pa = &a[0]` is het zelfde als `pa = a`.

In plaats van `a[i]` kunnen wij ook `*(a+i)` gebruiken.

De C compiler vertaalt `a[i]` intern naar `*(a+i)`.

Pointers en vectoren

Als een vector naam aan een functie doorgegeven wordt, wordt het adres van de vector doorgegeven.

voorbeeld:

```
int strlen( char *s)    /* lengte van een string */
 {
  int n;

  for ( n=0; *s != '\0' ; s++ )
    n++;
  return (n);
 }
```

s is een variabele en kan verhoogd worden.

s++ heeft geen effect op de string in de roepende functie.

U kunt volgende oproepen gebruiken:

```
streln("dit is een string van tekens");  /* een constante */
strlen(array);                           /* char array[100]; */
strlen(ptr);                             /* char *ptr; */
```

example23.c

```
#include<stdio.h>
#include<ctype.h>
#define SIZE 10

/* read next integer value from stdin and copy value to *pn */
int getint(int *pn)
 {
  int c,sign;

  while(isspace(c=getchar()))    /* ignore space */
   ;

  if ( !isdigit(c) && c != EOF && c !='+' && c !='-')  /* not a number */
   {
    return(0);
   }

  sign = (c=='-' ? -1 : 1);
  if (c=='+' || c=='-')
   {
    c=getchar();
   }
  for(*pn=0; isdigit(c); c=getchar() )
   {
    *pn=10 * *pn + (c-'0');
   }
  *pn *= sign;
  return(c);
 }
```

```
int main()
 {
  int n, array[SIZE];

  for(n=0; n < SIZE && getint(&array[n]) != EOF ; n++)
   ;

  for(n=0; n<SIZE; n++)
   {
    printf("%i ",array[n]);
   }
  printf("\n");
 }
```

Pointers en vectoren

```
int main()
 {
   int a[10];
   int *pa;

   pa = a;
   pa++;
   a = pa;
   a++;
 }
```

Welke toewijzingen zijn OK?

	OK	fault
pa = a;	(X)	( )
pa++;	(X)	( )
a = pa;	( )	(X)
a++;	( )	(X)

pa is een pointer

a is een vector

a wijst altijd naar &a[0] en kan niet verandert worden

Pointers en vectoren

```
int print_iets(char *s)
 {
   printf("%s \n", s);
   return(0);
 }

int main()
 {
   char text[]="Dit is een string van tekens";
   char *t;

   t = text;
   print_iets(t+4);
 }
```

de pointer wijst naar text[4]

Wat wordt afgedrukt?

A) Dit is een string van tekens

B) is een string van tekens

char pointers en functies

Een string constant zoals “Ik ben een string“ is een vector van tekens. De vector wordt door ‘\0‘ beëindigd. De lengte van de vector in het geheugen is om 1 groter dan de aantal tekens.

voorbeeld: printf(“een tekst \n“);
printf krijgt een pointer naar deze string.

```
char *pmessage;
pmessage= “een tekst \n“;
printf(“%s“,pmessage);
```

pmessage is een pointer naar de string. De inhoud wordt niet gekopieerd. C heft geen operator die een string als eenheid behandeld.

De functie strcpy kopieert de inhoud van een string.

De functie strcpy() kopieert de inhoud van de vector t naar de vector s.
s = t is een toewijzing van pointers, niet van de inhoud.

strcpy is een standaard functie (#include <string.h>)

twee voorbeelden voor deze functie (in plaats van de standaard functie):

```
void strcpy(char *s, char *t)   /* vector */
  {
    int i;

    i=0;
    while( (s[i] = t[i]) != '\0' )
      i++;
  }
```

```
void strcpy(char *s, char *t)  /* pointers */
  {
    while( (*s = *t) != '\0')
      {
        s++;
        t++;
      }
  }
```

Een goede programmeur doet dit zo:

```
void strcpy(char *s, char *t)
  {
    while ( (*s++ = *t++) != '\0') ;
  }
```

of nog korter:

```
void strcpy(char *s, char *t)
  {
    while (*s++ = *t++)  ;
  }
```

Initialisatie van vectoren

```
char *text;
text = “Dit is een tekst“;
```

Dit is geen toewijzing van de string naar text.
Dit gaat in C niet.

De pointer `text` wijst naar de locatie in het geheugen waar de string staat.

Initialisatie van vectoren

```
#include <stdio.h>

/* name of the month in readable form */
char *month_name(int month)
 {
  char *name[] = {
   “illegal month“,
   “January“,“February“,“March“,“April“,“May“,“June“,
   “July“,“August“,“September“,“October“,“November“,“December“ };

  return(  (month<1 || month>12) ? name[0] : name[month] );
 }

int main()
 {
  printf(“%s \n“, month_name(2) );
 }
```

Command line parameters

Bij oproep van een programma in Linux kunnen wij parameters doorgeven.
voorbeeld:
./taak9 123 456

De functie int main() heeft 2 parameters:

int argc	de aantal parameters
char *argv[]	een vector van pointers naar strings, de parameters
argv[0]	is de naam van het programma
argv[argc]	is een NULL pointer

```
int main(int argc, char *argv[] )
 {
  int i;

  for(i=1; i<argc; i++)
     printf("%s%s", argv[i], (i<argc-1) ? " " : "\n");

  return(0);
 }
```