Hoofdstuk 7: Input en output

#include <stdio.h> De standaard input en output library.

int getchar(void) leest een teken van standaard input.
Het einde van de input wordt door EOF (end of file) gegeven.

In Linux kunnen wij de input omleiden
prog < filename <enter>

“< bestand naam“ is niet deel van de argumenten in argv
<ctrl>D geeft EOF van de toetsenbord.

int putchar(int) geeft een teken op standaard output uit.

Standaard output kunnen wij met > omleiden.
prog > bestand <enter>

Geformatteerd output: printf

int printf(char *format, arg1, arg2, ...)
geeft interne waarde geformatteerd uit.

Tussen % en een conversie teken kunnen de volgende tekens staan

“-“ de veld wordt naar links uitgelijnd.
een cijfer geeft de minimaal aantal tekens aan.
“.“ aantal tekens voor en naar de komma.
“h“ voor short variabelen
“l“ voor long variabelen

int sprintf(char *string, char *format, arg1, arg2, ...)
In plaats van standaard output wordt het resultaat naar *string geschreven.

conversie tekens

d, i Type signed int.
o Type unsigned int printed in octal.
u Type unsigned int printed in decimal.
x Type unsigned int printed in hexadecimal as dddd using a, b, c, d, e, f.
X Type unsigned int printed in hexadecimal as dddd using A, B, C, D, E, F.
f Type double printed as [-]ddd.ddd.
e, E Type double printed as [-]d.dddedd where there is one digit printed before the decimal (zero only if the value is zero). The exponent contains at least two digits. If type is E then the exponent is printed with a capital E.
g, G Type double printed as type e or E if the exponent is less than -4 or greater than or equal to the precision. Otherwise printed as type f. Trailing zeros are removed. Decimal point character appears only if there is a nonzero decimal digit.
c Type char. Single character is printed.
s Type pointer to array. String is printed according to precision (no precision prints entire string).
p Prints the value of a pointer (the memory location it holds).
n The argument must be a pointer to an int. Stores the number of characters printed thus far in the int. No characters are printed.
% A % sign is printed.

Geformatteerd input: scanf

int scanf(char *format, arg1, arg2, ...)
leest tekens van standaard input.

int sscanf(char *string, char *format, arg1, arg2, ...) leest geformatteerde waarde van een string.

Tussen % en een conversie teken kunnen de volgende tekens staan
-h voor short typen (short int ...)
-l voor long typen (long int ...)
f -> float lf -> double

Bestanden - input en output

FILE *fp; /* een file pointer */
FILE *fopen(char *name, char *mode)
  <mode> heft volgende waarde:
  “r“ om een bestand te lezen
  “w“ om een bestand te schrijven
  “a“ om iets aan een bestaande bestand toe te voegen.

int getc(FILE *fp)
leest de volgende teken uit de bestand fp.
De return waarde is EOF (end of file) aan het eind van de bestand.

int putc(int c, FILE *fp)
schrijft een teken in de bestand fp.

Voor geformatteerd input en output hebben wij de functies
int fscanf(FILE *fp, char *format, ...)
int fprintf(FILE *fp, char *format, ...)

int fclose(FILE *fp)

stdin en stdout zijn speciale FILE pointers, zij wijzen naar standaard input en output.

Geformatteerd input en output

stdin/stdout	printf	scanf
file	fprintf	fscanf
string	sprintf	sscanf

Input en output van regels.

char *fgets(char *line, int maxline, FILE *fp)
leest maximaal maxline tekens van de bestand fp.

int fputs(char *line, FILE *fp)
schrijft een regel <line> naar het bestand fp.

Uitvoeren van commando's

De functie system(char *s) voert een commando op het (Linux) systeem uit.

Geheugenbeheer - malloc en free

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

#define MAX 100

int doe_iets(char **names){}
```

```
int main()
 {
  int i=0;
  char *names[MAX];
  char text[80];
  FILE *fp;

  for(i=0; i<MAX; i++)
   names[i]=NULL;

  fp=fopen("filename","r");
  while ( fgets(text,80,fp) != NULL )
   if( i<MAX )
     {
      names[i] = malloc(strlen(text));
      strcpy(names[i],text);
     }
  fclose(fp);

  doe_iets(names);

  for(i=0; i<MAX; i++)
   free(names[i]);
 }
```

Geheugenbeheer

De functie
void *malloc(size_t n)
wijst n bytes in de geheugen aan een pointer toe.

De functie
void free(void *)
geeft de ruimte in de geheugen weer vrij.
***Let op*: alleen pointers die met malloc geinitialiseerd worden zijn kunnen met free weer vrij gegeven worden.**

Standaard C bibliotheek

In de standaard C bibliotheek zijn veel handige functies:
De belangrijkste zijn:
ctype.h
math.h
stdio.h
stdlib.h
string.h
time.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

ctype.h

isalnum	checks if a byte is alphanumeric
isalpha	checks if a byte is alphabetic
islower	checks if a byte is lowercase
isupper	checks if a byte is an uppercase byte
isdigit	checks if a byte is a digit
isxdigit	checks if a byte is a hexadecimal byte
iscntrl	checks if a byte is a control byte
isgraph	checks if a byte is a graphical byte
isspace	checks if a byte is a space byte
isprint	checks if a byte is a printing byte
ispunct	checks if a byte is a punctuation byte
tolower	converts a byte to lowercase
toupper	converts a byte to uppercase

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

from Wikipedia

stdlib.h

```
double atof(const char *str)  string to double (NOT float)
int atoi(const char *str)     string to integer
atol                          string to long
strtod                        string to double
strtol                        string to long int
strtoul                       string to unsigned long int
strtoll                       string to long long int
strtoull                      string to unsigned long long int

int rand(void)
        generates a pseudo-random number
void srand(unsigned int seed)
        set the rand() pseudo-random generator seed [common convention
        uses time() to seed]

malloc   allocate memory from the heap
free     release memory back to the heap

void exit(int status)                 terminate program execution
int system(const char *command)       execute an external command
```

string.h

```
char *strcat(char *dest, const char *src);
        appends the string src to dest
char *strncat(char *dest, const char *src, size_t n);
        appends at most n bytes of the string src to dest
char *strchr(const char *, int c);
        locates byte c in a string, searching from the beginning
char *strrchr(const char *, int c);
        locates byte c in a string, searching from the end
int strcmp(const char *, const char *);
        compares two strings lexicographically
int strncmp(const char *, const char *, size_t n);
        compares up to the first n bytes of two strings lexicographically
char *strcpy(char *dest, const char *src);
        copies a string from one location to another
char *strncpy(char *dest, const char *src, size_t n);
        write exactly n bytes to dest, copying from src or add 0's
size_t strlen(const char *);
        finds the length of a C string
char *strstr(const char *haystack, const char *needle);
        finds the first occurrence of the string "needle" in the longer string
        "haystack"
```

time.h

CLOCKS_PER_SEC Clock ticks per second (macro)
NULL Null pointer (macro)

types

clock_t Clock type (type)
size_t Unsigned integral type (type)
time_t Time type (type)
struct tm Time structure (type)

Time manipulation
clock_t clock(void); Clock program sec = clock() / CLOCKS_PER_SEC
double difftime(time_t, time_t); Return difference between two times
time_t mktime(struct tm *); Convert tm structure to time_t
time_t time(time_t *); Get current time

Conversion:
char *asctime(const struct tm *); Convert tm structure to string
char *ctime(const time_t *); Convert time_t value to string
struct tm *gmtime(const time_t *); Convert time_t to tm as UTC time
struct tm *localtime(const time_t *); Convert time_t to tm as local time
size_t strftime(char *, size_t, const char *, const struct tm *); Format time as string

Grafische representatie van data

Jullie zijn nu klaar om taken in wiskunde of natuurkunde met de hulp van een programma op te lossen.
Bijvoorbeeld hebben wij vaak in het natuurkunde practicum de opdracht een aantal metingen te verwerken.

voorbeeld: Wij hebben een tabel met metingen:
10 1.34
20 2.58
30 5.25
40 6.87
Die schrijven wij in de bestand data.in.

In een C programma lezen wij dit bestand en wij doen de nodige berekeningen.
Daarnaar roepen wij gnuplot om de data grafisch weer te geven.

Grafische representatie van data

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

/* graphical output met gnuplot */

/* this functions reads the data and performs necessary
calculations etc. */
int read_data(char *name)
 {
  FILE *fp;
  if( (fp=fopen(name,"r"))==NULL )
    {
     printf("file ERROR!\n");
     exit(1);
    }
  ...
  fscanf( ...);
  ...
  close(fp);
  return(0);
 }
```

```
/* this functions writes the data to disk */
int write_data(char *name)
 {
  ...
  fprintf( ... );
  ...
  return(0);
 }

int main()
 {
  char input[]="data.in";
  char output[]="data.out";

  read_data(input);

  write_data(output);

  /* call gnuplot to display data */
  system("gnuplot commands");
 }
```

Grafische representatie van data

GNU plot (http://gnuplot.info)

gnuplot is een standaard programma op onze Linux systeem.

> ssh -X username@lilo.science.ru.nl
> gnuplot

De data zijn in de bestand “data.out“.

> plot “data.out“ with lines

De data worden in een raam uitgegeven.

> gnuplot commands

gnuplot wordt uitgevoerd. “commands“ is een bestand met commando‘s voor gnuplot.

Herhaling

Hoofdstuk 2: Data types, operators en toewijzingen

Variabele namen

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

Data types en lengte in de geheugen

char	**1 byte**
int	**4 bytes**
float	**4 bytes**
double	**8 bytes**

Variaties van deze basistypen zijn: short en long

Bovendien signed en unsigned voor elke char en int type.
voorbeeld: char: -128 .. 0 .. 127 en unsigned char 0 .. 255
8 bit -2^7 .. 2^7-1 0 .. 2^8-1

Constanten

Constanten worden met #define gedefinieerd.
voorbeeld
#define LENGTH 80 /* een integer waarde */

Ook een variabel kan als een constante gedefinieerd worden
const double e = 2.71828182845905;

Niet-afdrukbare tekens

\\ Backslash character.
\? Question mark character.
\' Single quotation mark.
\" Double quotation mark.
\a Audible alert.
\b Backspace character.
\e <ESC> character. (This is a GNU extension.)
\f Form feed.
\n Newline character.
\r Carriage return.
\t Horizontal tab.
\v Vertical tab.
\o, \oo, \ooo Octal number.
\xh, \xhh, \xhhh, ... Hexadecimal number.

Rekenkundige operatoren

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

Vergelijke en logische operatoren

Vergelijkingsoperatoren:
> >= < <= hogere prioriteit
== != legere prioriteit

Let op a = b toewijzing
a==b vergelijking

logische operatoren: && en || of

! 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 )
```

Increment en decrement operator

De operatoren ++ -- kunnen voor of naar een variabel 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

Bit manipulatie

& logisch en 0x0f & 0xff --> 0x0f
| logisch of 0x0f | 0xff --> 0xff
^ exclusief of 0x0f ^ 0xff --> 0xf0
<< bit verschuiving naar links 00001111 << 2 --> 00111100
x << 2 --> x = x*4
>> bit verschuiving naar rechts
~ bit complement (unitair) -0x0f --> 0xf0

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

voorwaardelijke toewijzingen

(expr1 ? expr2 : expr3)
Teerst wordt expr1 berekend. Als dit niet 0 is wordt expr2 berekend, anders expr3.

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 ;
Instructies worden door de haakjes {} in een block samengevat.

if else

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

else if

```
if ( expression)
   statement;
else if (expression)
   statement;
else if (expression)
   statement;
else
   statement;
```

switch

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

Lussen (loops) while en for

```
while (expression)
  statement

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

Lussen (loops) do while

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

break en continue

Met **break** wordt een lus (for, while en do) vroegtijdig **beeindigt**.

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

Hoofdstuk 4: functies en programmastructuur

functies

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

Definitie bestanden (header files)

Standaard functies zijn in bibliotheken (libraries) samengevat.
header files worden met #include <filename> in een programma ingevoegd.

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));
```

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);
 }
```

Hoofdstuk 5: pointers en vectoren

Pointers en adressen

De adres-operator geeft de adres van een object in de geheugen.
p = &c;

De inhoud-operator geeft de inhoud van een pointer.
a = *p;

voorbeeld:
int *ip; /* pointer naar int *
int *ip is een nieuwe type van variabel, een pointer.

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 wisselen.

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

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

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

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

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] ...
```

```
int *pa;
pa = &a[0];
pa = a;
```

***(a+i) --> a[i]**

Adressen rekenkunde (pointer arithmetic)

In #include<stdio.h> is de NULL pointer gedefinieerd.
pa = NULL; wijst naar “nergens“.

p+n wordt aan de variabele type aangepast.
p++ voor een char *p bedoeld een verhoging om 1. sizeof(char)=1
p++ voor een int *p bedoeld een verhoging om 4. sizeof(int)=4

Toegestaan operaties met pointers zijn:
-toewijzing van objecten van de zelfde type p = q
-+ en - van een waarde p++, p--, p += 3, p -=4
-toewijzing en vergelijk met de NULL pointer
-vergelijk van twee pointers die naar elementen van de zelfde vector wijsen
 p < q

Meerdimensionale vectoren

C heft meerdimensionale rechthoekige vectoren.
int a[10][20]; /* lijn kolom */

In C is een tweedimensionaal vector een eendimensionaal vector en de elementen van deze vector zijn weer een vector.

Daarom schrijven wij a[10][20] en NIET a[10,20]

Command line parameters

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

De functie int main() heft 2 parameters:
int argc de aantal parameters
char *argv[] een vector van pointers naar strings, de parameters
argv[0] is de naam van de 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);
 }
```

Hoofdstuk 6: structures

Een structuur (structure) is een collectie van variabelen. Met structuren kunnen wij groepen van variabelen vormen. Dit is nuttig in grote programma's.

struct <name> {
<variabelen>
}
definieert een structuur (structure). De <variabelen> zijn de "members" van dit structuur.
Een structuur is een nieuwe type variabele.

structuren en functies

voorbeeld

```
/* makepoint: generate a point from x and y coordinates */
struct point makepoint(int x, int y)
 {
  struct point temp:

  temp.x = x;
  temp.y = y;
  return(temp);
 }
```

pointers naar structuren

Pointers naar structuren werken zoals pointers naar gewoon variabelen.

```
struct point origin, *pp;
pp = &origin;
printf(„origin is (%i,%i) \n“, (*pp).x, (*pp.y) );
```

Pointers naar een structuur worden zeer vaak gebruikt. Daarom hebben wij in C en speciale manier voor dit constructie
Als p een pointer naar een structuur is dan wijst
p -> component naar de component in de structuur.

(*pp).x pp->x

De selectie operatoren “.“ en “->“ hebben tezamen met de haakjes voor functies () en arrays [] de hoogste prioriteit.

typedef

Nieuwe type namen kunnen in C met typedef gedefinieerd worden.

```
typedef struct {
    char *voornaam;
    char *achternaam;
} Student;
```

De nieuwe type naam Student staat voor een structuur.

typedef genereert geen nieuwe type. Met typedef wordt een nieuwe naam voor een bestaande type of structuur definieert.
Maar de gebruik van typedef maakt een programma beter lesbaar.

Hoofdstuk 7: Input en output

#include <stdio.h> De standaard input en output library.

int getchar(void) leest een teken van standaard input.
Het einde van de input wordt door EOF (end of file) gegeven.

In Linux kunnen wij de input omleiden
prog < filename <enter>

“< bestand naam“ is niet deel van de argumenten in argv
<ctrl>D geeft EOF van de toetsenbord.

int putchar(int) geeft een teken op standaard output uit.

Standaard output kunnen wij met > omleiden.
prog > bestand <enter>

Geformatteerd output: printf

int printf(char *format, arg1, arg2, ...)
geeft interne waarde geformatteerd uit.

Tussen % en een conversie teken kunnen de volgende tekens staan

“-“ de veld wordt naar links uitgelijnd.
een cijfer geeft de minimaal aantal tekens aan.
“.“ aantal tekens voor en naar de komma.
“h“ voor short variabelen
“l“ voor long variabelen

int sprintf(char *string, char *format, arg1, arg2, ...)
In plaats van standaard output wordt het resultaat naar *string geschreven.

conversie tekens

d, i Type signed int.
o Type unsigned int printed in octal.
u Type unsigned int printed in decimal.
x Type unsigned int printed in hexadecimal as dddd using a, b, c, d, e, f.
X Type unsigned int printed in hexadecimal as dddd using A, B, C, D, E, F.
f Type double printed as [-]ddd.ddd.
e, E Type double printed as [-]d.dddedd where there is one digit printed before the decimal (zero only if the value is zero). The exponent contains at least two digits. If type is E then the exponent is printed with a capital E.
g, G Type double printed as type e or E if the exponent is less than -4 or greater than or equal to the precision. Otherwise printed as type f. Trailing zeros are removed. Decimal point character appears only if there is a nonzero decimal digit.
c Type char. Single character is printed.
s Type pointer to array. String is printed according to precision (no precision prints entire string).
p Prints the value of a pointer (the memory location it holds).
n The argument must be a pointer to an int. Stores the number of characters printed thus far in the int. No characters are printed.
% A % sign is printed.

Geformatteerd input: scanf

int scanf(char *format, arg1, arg2, ...)
leest tekens van standaard input.

int sscanf(char *string, char *format, arg1, arg2, ...) leest geformatteerde waarde van een string.

Tussen % en een conversie teken kunnen de volgende tekens staan
-h voor short typen (short int ...)
-l voor long typen (long int ...)
f -> float lf -> double

Bestanden - input en output

FILE *fp; /* een file pointer */
FILE *fopen(char *name, char *mode)
  <mode> heft volgende waarde:
  “r“ om een bestand te lezen
  “w“ om een bestand te schrijven
  “a“ om iets aan een bestaande bestand toe te voegen.

int getc(FILE *fp)
leest de volgende teken uit de bestand fp.
De return waarde is EOF (end of file) aan het eind van de bestand.

int putc(int c, FILE *fp)
schrijft een teken in de bestand fp.

Voor geformatteerd input en output hebben wij de functies
int fscanf(FILE *fp, char *format, ...)
int fprintf(FILE *fp, char *format, ...)

int fclose(FILE *fp)

stdin en stdout zijn speciale FILE pointers, zij wijzen naar standaard input en output.

Input en output van regels.

char *fgets(char *line, int maxline, FILE *fp)
leest maximaal maxline tekens van de bestand fp.

int fputs(char *line, FILE *fp)
schrijft een regel <line> naar het bestand fp.

Uitvoeren van commando‘s

De functie system(char *s) voert een commando op het (Linux) systeem uit.

Geheugenbeheer

De functie
void *malloc(size_t n)
wijst n bytes in de geheugen aan een pointer toe.

De functie
void free(void *)
geeft de ruimte in de geheugen weer vrij.
***Let op*: alleen pointers die met malloc geinitialiseerd worden zijn kunnen met free weer vrij gegeven worden.**

Standaard C bibliotheek

In de standaard C bibliotheek zijn veel handige functies:
De belangrijkste zijn:
ctype.h
math.h
stdio.h
stdlib.h
string.h
time.h

Programmeren 1
Programming 1

2012/2013

NP033B
1e jaar, kwartaal II, 3 ec

TENTAMEN

Woensdag, 23 Januari 2013
12:30 - 15:30
Gymnasion Hal 2

Succes!