Inleiding Programmeren in C
(onder labwindows)

Contact: Ivo van Vulpen & Marcel Vreeswijk

versie 0
op www: www.nikhef.nl/~h73

Contents

1 Inleiding **4**

1.1 Materiaal . . . 4
1.2 Computers en accounts . . . 4
1.3 Werkwijze . . . 4
1.3.1 Specifiek voor Labwindows . . . 5
1.3.2 In den beginne: "Mijn eerste C-programma" . . . 6
1.4 Iets meer dan het minimum . . . 7
1.5 Basis elementen van de taal C . . . 8
1.5.1 De character set . . . 8
1.5.2 Constanten . . . 8
1.5.3 Symbolische namen . . . 9
1.5.4 Variabelen . . . 9
1.6 Arithmetische expressies . . . 9
1.7 Arithmetisch assignment . . . 10
1.8 Uitvoer via 'printf' . . . 11
1.9 Invoer via 'scanf" . . . 12
1.10 Mathematische functies . . . 13
1.11 Losse opmerkingen . . . 14
1.12 Opdrachten . . . 14
1.12.1 Opdracht 1 . . . 14
1.12.2 Opdracht 2 . . . 14
1.12.3 Opdracht 3 . . . 15

2 Control statements **15**

2.1 Het if-statement . . . 15
2.1.1 Relationele en logische expressies/voorwaarden . . . 16
2.1.2 De eentakkige if . . . 16
2.1.3 De tweetakkige if . . . 16
2.1.4 De meertakkige if . . . 17
2.2 De for-loop . . . 17
2.2.1 Numeriek integreren . . . 18
2.2.2 Voorbeeld van een for-loop . . . 18
2.2.3 Arrays van variabelen . . . 19
2.3 De while-loop en de do-while-loop . . . 20
2.4 Grafische weergave van resultaat . . . 21
2.5 Opdrachten . . . 22
2.5.1 Opdracht 4 : het if statement : $ax^2 + bx + c = 0$ . . . 22
2.5.2 Opdracht 5 : arrays en het plotpakket . . . 23

3 Gebruik van functies in C **23**

3.1 Inleiding . . . 23
3.2 Het nut van functies . . . 24
3.3 Functies zonder parameter en zonder return-waarde . . . 24

3.4 Functies met parameter en zonder return-waarde . . . 26
3.4.1 Opdracht 6 . . . 27
3.5 Functies met parameter en met return-waarde . . . 27
3.6 Array als parameter . . . 29
3.7 Functie resultaten in een array-parameter . . . 31
3.8 Functieresultaten in een (enkelvoudige) parameter . . . 33
3.8.1 Opdracht 7 . . . 34
3.9 Communicatie via globale variabelen . . . 35
3.9.1 Lokale variabelen . . . 35
3.9.2 Globale variabelen . . . 36

4 Nog enkele 'handige zaken' 37
4.1 Gebruik van #define . . . 37
4.2 Het gebruik van de 'break'-opdracht . . . 38
4.2.1 Opdracht 8 . . . 39

5 LabWindows specifiek 39

1 Inleiding

Een computertaal kennen, programmeren dus, is een vaardigheid die voor de meeste natuurkundigen een noodzaak is. Naaast numeriek natuurkundige problemen doorrekenen worden ook instrumenten aangestuurd met behulp van computers. Deze cursus in de programmeertaal C is een voorbereiding op beide. De instrumentbesturing is een onderdeel van het praktikum dat volgt op deze cursus en 'Numerieke Natuurkunde' is een vervolgcursus in het tweede jaar. Als we nog verder denken, komen we in het 3de jaar terecht, of zelf de masteropleiding, voor sommige studierichtingen zal je werken met grote software-pakketten in andere programmeertalen, zoals C++, Java of Python. Toch zul je veel aan de je C kennis hebben omdat dit de basis van al deze programmeertalen is. Tot slot moet opgemerkt worden dat programmeren een goede training is voor je hersenen om eens abstract te denken en leer je uiteindelijk om problemen te vertalen naar wiskundige algoritmes.

Voor deze cursus werken we in de Labwindows omgeving, maar we houden het zo algemeen mogelijk. De vervolgcursus instrumentbesturing zal dieper ingaan op het besturen van interfaces waarvoor de Labwindow omgeving uitermate geschikt is.

1.1 Materiaal

Naast deze syllabus is er studiemateriaal en voorbeelden te vinden op de website:
http://www.nikhef.nl/~h73/ccursus.html

- Als U een uitgebreider *boek* over C zoekt, is het volgende geschikt :
  Al Kelley en Ira Pohl : "A Book on C : Programming in C"
- Technischer, maar zeker de moeite waard: *The C Programming language* door B. Kernighan en D. Ritchie.
- Er is veel informatie over C beschikbaar op het web. Als je zoekt naar een bepaald onderwerp, bijvoorbeeld 'sinus functie', kun je in een zoekmachine even zoeken naar 'ANSI C sinus' en je krijgt meteen talloze voorbeelden. Overigens staat ANSI voor 'American National Standards Institute'; dit is de standaard waaran we ons zoveel mogelijk aan zullen houden.

1.2 Computers en accounts

We werken voor deze cursus op de computers van de UvA waarvoor jullie allemaal al een account hebben. Werk zo mogelijk iedere sessie op dezelfde computer en meldt als er iets niet werkt. Start het programma Labwindows dat onder het startmenu beschikbaar is, waarschijnlijk onder de firma-naam 'National Instruments'.

1.3 Werkwijze

De werkwijze bij deze cursus, waarbij U *zelf* programma's schrijft en laat uitvoeren, bevat de volgende *basiselementen* :

- U moet de door U gemaakte programma-*code* 'intikken' met behulp van een *tekst-editor*. De *file* die aldus ontstaat bevat Uw **source-code**.
- Deze code moet worden *vertaald* door een **C-compiler**. Deze compiler controleert ook of Uw programma aan de *spelregels* van de **taal C** voldoet. De *file* die aldus ontstaat bevat de **object-code**. De object code wordt niet altijd als aparte file opgeslagen, maar is dan tijdelijk aanwezig in het computer geheugen.
- Vervolgens moet de *object-code* worden *omgezet* in een vorm die *in werking* kan worden gesteld (*executeerbaar* is). Deze file heet de **executable**.
- Ten slotte moet de *executable* in werking worden gezet. De resultaten van Uw programma zullen moeten worden *geprint*, of *geplot* als het om *grafische* resultaten gaat.

Wat betekent dit nu allemaal praktisch? In de Labwindows omgeving zijn al deze stappen onder de verschillende menu's beschikbaar en daar gaan we in de volgende sectie gebruik van maken.

1.3.1 Specifiek voor Labwindows

Labwindows is ontworpen om grote programma's te maken die weer kunnen bestaan uit verschillende projecten en dat brengt enige overhead met zich mee. Daar willen we niet te lang bij stil staan, want we willen gewoon beginnen.

Begin met een bestaande workspace voor deze cursus De makkelijkste methode om te beginnen is om een zogenaamde workspace te kopieren die voor deze cursus is klaargezet in de directory 'labwindows' via de link onder:

http://www.nikhef.nl/~h73/ccursus.html

vervolgens de directory 'labwindows' kopieren vanuit je browser (aanklikken en save-as naar je eigen lokale disc). Als dat niet lukt, kun je het per file (of alle files selectie) proberen. Dus eerst een directory maken op je lokale disc en vervolgens kopieren uit 'labwindows' (de directories hoef je niet te kopieeren).

Open de workspace (door via je PC op 'voorbeeld.cws' te klikken) of door Labwindows te openen en dan onder 'File->Open' de 'voorbeeld.cws' workspace te openen. In de werkruimte staan verschillende projecten waar een eigen C file aanhangt. Als je een bepaald voorbeeld in een project wilt bekijken kun je op de file klikken. Als je op het icoon van het project met de rechtermuis klikt krijg je het menu voor het project. Voor straks is het belangrijk om het eerste project 'hello' actief te maken, dat kan o.a. via 'File->Set Active Project'.

Helemaal vanuit het niets beginnen Dat doe je het snelst door onder het menu 'File->New' een nieuwe 'workspace' te maken. Er wordt meteen ook een project in de workspace gemaakt. Je kunt ook zelf nog een een nieuw project beginnen dat ook onder het menu 'File->New' te vinden is. Save nu eerst de workspace en je nieuwe project(en); dat is belangrijk, want als alles 'Untitled' heet zie je door de bomen het bos niet meer. Je

kunt ook het icoon van je project aanklikken en met de rechterklik meer opties krijgen, bijvoorbeeld om het project te Saven of om de title van het project kunt veranderen (die hoeft weer niet hetzelfde te zijn als de file-naam, dat is een beetje verwarrend).

Vervolgens wil je een file openen waarin je C-code getyped kan worden. Je raad het al: wederom onder 'File->New'. Save de file onder een goede naam. Voer al deze stappen uit. De C file hoort onder een project te te staan. Als dat niet vanzelf is gebeurd, was het project niet actief. Als je meerdere projecten in je workspace hebt gedefinieerd moet je altijd opletten welk project actief is. Dat kun je doen onder 'File->Set Active Project'. Vervolgens kun je aan het project de C-file toevoegen door met rechtermuis op het project te klikken en dan 'Add File'. Een project kan meerdere files bevatten, maar dat hoeft niet.

Een volledige introductie Labwindows kun je vinden op
http://www.nikhef.nl/~h73/docu_labwindows.pdf
maar in deze cursus gaat het ons met name om het programmeren in C.

1.3.2 In den beginne: "Mijn eerste C-programma"

In deze sectie ga je je eigen een programma maken, waarna je dit programma kunt compileren met menu 'Compile' en kunt uitvoeren met 'Run'. Zoek deze menu's alvast maar even. Verder gaan we ervan uit dat je weet hoe je files kunt opslaan en weer openen op de gebruikelijke manier zoals o.a. in de microsoft producten Word en Powerpoint. Als je dat nog niet kunt, moet je dat maar even aan je buurman vragen en uiteraard zorgen dat je Word/Powerpoint zo snel mogelijk leert.

Een minimaal C programma ziet er als volgt uit:

```
// Begin met een goede beschrijving van je programma.
// Noteer hier ook je naam en de datum.
int main (void)
{
// Geef commentaar bij ieder belangrijk gedeelte van je programma!

    return 0 ;

}
```

- Dit programma *doet niks*, *maar* :
- De C-compiler zal er *niet* tegen protesteren, omdat de *vorm* correct is :
  - Er komt een *hoofd*-programma in voor, via

    `int main (void)`
  - Het hoofdprogramma wordt netjes afgesloten, via de terugkeer-opdracht

    `return 0 ;`

    Merk op dat deze opdracht wordt afgesloten met een *punt-komma*

  - De (afsluitende) *code* is *netjes ingebed* tussen

    ```
    {
    }
    ```

- Elk programma moet dus *in elk geval* bovenstaande vorm hebben.

Een eerste poging voor een klein C-programma, maar nog steeds minimaal, zie je in de file 'hello.c', klik daar maar op of type het onderstaande over:

```
 #include <ansi_c.h>   // maakt de standaard bekend in labwindows.
//  Buiten labwindows zie je vaak #include ‘‘stdio.h’’.
// Mijn eerste programma in Labwindows....
int main(void) {
 char s;  // declareer een karakter s voor dummy input met getchar
//
// hier kun je van alles programmeren......
//
//--
// bijvoorbeel een printf
//--
printf("hello world \n");

// truukje om het output venster even open te houden
printf("type enter to continue");
s=getchar();
// geef een 0 terug aan het operating systeem.
return 0;
}
```

Vergeet niet het 'hello' project te activeren 'File->Set Active Project'. Vervolgens kun je de C code compileren met 'Build->Compile'. Als er fouten worden gevonden kijk je in het schermpje onderin naar de lijnregel en los je fouten op (begin altijd met de eerste en probeer het opniew, want soms zijn vele fouten het gevolg van een eerste simpele fout). Hierna kun je de code excuteren onder 'Run->Debug' (of direct via de groene pijl onder menu's). Als je echt wil leren debuggen kun je de labwindows manual gebruiken voor meer informatie; in deze cursus slaan we dit over. Verander het programma zodat je eigen naam wordt uitgeprint en executeer opnieuw. Je bent nu klaar voor meer!

1.4 Iets meer dan het minimum

Een voorbeeld van een programma dat *wel iets doet* is :

```
int main (void)
{
    int iks ;   // variable voor de berekening.
```

```
    iks = 1 ;
    iks = iks + 2 ;

    return 0 ;

}
```

Dit programma :

- Werkt met een *variabele* met de *naam* iks, *gedeclareerd* in

  ```
  int iks ;
  ```

- Kent eerst de waarde 1 toe aan de variabele iks, in

  ```
  iks = 1 ;
  ```

- Telt vervolgens 2 op bij de waarde van iks, en
  Slaat het resultaat op in de variabele iks, in

  ```
  iks = iks + 2 ;
  ```

- Merk op dat (bijna) alle regels worden afgesloten met een *punt-komma*

1.5 Basis elementen van de taal C

Bovenstaand programmaatje stelt ons in staat een aantal *basis- elementen* van de taal **C** te bespreken.

1.5.1 De character set

De **character** set , die gebruikt kan worden, bestaat uit

Kleine letters	a,b,———,z
Hoofdletters	A,B,———,Z
Cijfers	0,1,———,9
Underscore	_
Speciale tekens	spatie = + - / * enz.

1.5.2 Constanten

- De meest voorkomende vorm van **constanten** zijn gewoon *getallen*, zoals :

  ```
  1   2   5.73  12.3e9
  ```

  Dit zijn *arithmetische* constanten.

- Een andere voorkomende vorm is die van een *character string*, ingesloten tussen *dubbele quotes*, zoals :

  ```
  "Numeriek"  "1"
  ```

1.5.3 Symbolische namen

- Worden gebruikt ter aanduiding van:variabelen, programma-delen etc.
- Moeten bestaan uit alphanumerieke symbolen (letters of cijfers)
- Mogen underscores bevatten.
- Het eerste symbool moet een letter zijn.
- Er is verschil tussen hoofd- en kleine letters; de *conventie* is om *geen* hoofdletters te gebruiken.
- Voor de leesbaarheid van het programma is het van belang 'betekenisvolle' (en niet te lange) namen te kiezen,zoals
  dag, maand, jaar in plaats van a, i, j
- In ons voorbeeld is 'iks' een *voorbeeld* van een symbolische naam.
- Ook 'main' is trouwens een symbolische naam, hier (verplicht) gebruikt voor het *hoofd*-programma.

1.5.4 Variabelen

- Worden genoemd met een symbolische naam.
- Het **type** wordt bepaald in een **declaratie** statement.

int	iks	een geheel getal (integer)
float	snelheid	een reeel getal (gewone precisie)
double	snelheid	een reeel getal (hogere precisie)
char	letter	kan een character bevatten

- Voor *elke* variabele *moet* het type worden gedeclareerd. Dit moet in elk geval gebeuren *voordat* de variabele wordt *gebruikt*. Een goede conventie is, om dit te doen aan het *begin* van het programma-deel, waarin de variable wordt gebruikt.
- In ons programma declareren we het type van de variabele 'iks' via

```
int iks ;
```

1.6 Arithmetische expressies

- Een voorbeeld van een arithmetische expressie in ons programma is :

```
iks + 2
```

- In een arithmetische expressie worden (arithmetische) constanten en variabelen met elkaar 'gecombineerd' via (arithmetische) *operatoren*. Deze operatoren zijn :
  - $+$ optellen
  - $-$ aftrekken
  - $*$ vermenigvuldigen
  - $/$ delen
  - $\%$ bereken modulus
- (Andere) Voorbeelden zijn :

```
a + b / 2.
c + 3. * d
```

- Belangrijk is de *volgorde* waarin expressies worden verwerkt. De *regels* hiervoor zijn :
  - $*$ $/$ en $\%$ hebben *prioriteit* boven $+$ en $-$.
  - $*$ $/$ en $\%$ zijn *onderling gelijkwaardig*, dwz dat ze worden afgewerkt in de volgorde waarin ze voorkomen, werkend van links naar rechts.
  - $+$ en $-$ zijn onderling gelijkwaardig, en worden (ook) in volgorde afgewerkt.
  - deze *standaard*-volgorde kan worden gewijzigd door het gebruik van 'haakjes' $(\ )$. Het uitwerken van delen van expressies 'tussen haakjes" heeft voorrang.
  - voorbeelden zijn :

```
a + b / 2.       (a + b)  / 2.
c + 3. * d       (c + 3.) * d
d / 4. + a       d / (4. + a)
```

- Wanneer twee *integer* variabelen of constanten op elkaar worden *gedeeld* wordt het *resultaat* weer een integer (naar beneden afgerond). $1/6$ levert dus 0 als resultaat.

1.7 Arithmetisch assignment

- Voorbeelden van een arithmetisch assignment zijn :

```
iks = 1 ;
iks = iks + 2 ;
```

  Merk op dat beide *regels* worden afgesloten met een '*punt-komma*'.

- De vorm is : **variabele=expressie ;**
  Variabele en Expressie moeten **arithmetisch** zijn
- De **expressie** mag bestaan uit een **enkele** (arithmetische) constante of variabele, eventueel voorafgegaan door een $-$ teken.

```
iks = - 1   ;
iks = - iks ;
iks = - iks + 2 ;
```

- N.B. Een assignment is heel iets anders als een 'vergelijking'.
  iks =iks + 1 is dus prima.

1.8 Uitvoer via 'printf'

- In het voorbeeldprogramma wordt wel iets uitgerekend, maar het resultaat wordt op geen enkele wijze *zichtbaar* gemaakt. Hierin kan worden voorzien door het programma op de volgende wijze *uit te breiden* :

```
#include <stdio.h>

int main (void)
{
    int iks ;

    iks = 1 ;
    iks = iks + 2 ;

    printf(" de waarde van iks = %d \n", iks) ;
 // LET OP: %d voor integer waarden, %f voor float waarden.

    return 0 ;

}
```

- Bij executie van het programma zal op Uw scherm de volgende tekst verschijnen :

```
 de waarde van iks = 3
```

De 'printf'-opdracht is een (eerste) voorbeeld van *aanroep van een functie*.

- de *naam* van de functie is 'printf'
- de functie wordt aangeroepen met *twee parameters*, van elkaar gescheiden door een 'komma'.
- U hoeft deze functie niet zelf te 'schrijven' : hij is beschikbaar in een *bibliotheek* van de C-compiler.
  de naam van deze bibliotheek is 'stdio' - de definities van de functies in deze bibliothees staan in de header-file 'stdio.h'
- U moet *wel* in Uw programma aangeven dat U deze bibliotheek wilt gebruiken. Dit gebeurt aan het begin van het programma, via

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

- de eerste parameter in de aanroep van 'printf' is een *character string* (ingesloten tussen 'dubbele quotes' " ").
  de tweede variable is de *naam* van een *variable*.
  - de tekst ' de waarde van iks = ' wordt gewoon uitgeprint.
  - de %d is een *format descriptor*. U geeft er mee aan dat 'op deze plaats' de *waarde van een variabele* moet worden uitgeprint.
    welke variabele dat is, wordt aangegeven in de tweede parameter.
  - met \n wordt aangegeven dat na deze print moet worden gestart op een *nieuwe regel*
- er zijn verschillende *format descriptors*.
  - %d wordt gebruikt voor het printen van *gehele getallen*.
    het aantal printposities dat nodig is, wordt bepaald door de waarde van de variabele.
    met bijv. %10d geeft U *zelf* aan dat er 10 printposities moeten worden gereserveerd.
  - %f wordt gebruikt voor het printen van *reele getallen*.
    U heeft *geen controle* over het totale aantal printposities, en ook niet over het aantal decimalen.
    met bijv %10f reserveert U in totaal 10 printposities voor het getal; U heeft geen controle over het aantal decimalen.
    met bijv %10.3f reserveert U in totaal 10 printposities voor het getal, en U vraagt om drie decimalen.

1.9 Invoer via 'scanf"

Het is mogelijk om vanuit Uw programma opdracht te geven om getalwaarden van variabelen in te lezen'vanaf het toetsenbord'. Een voorbeeld vind je in het volgende programma of in de file 'ex1.c' van de Workspace.

```
#include <stdio.h>

int main (void)
{
    float a, b, c ;

    printf(" geef de waarden van de getallen a,b,c\n");
    scanf ("%f %f %f", &a, &b, &c);

    printf(" de ingetikte waarden van a b c zijn : %f %f %f\n",a,b,c);
```

```
    return 0 ;

}
```

- het eerste deel van de de scanf-opdracht geeft aan dat er drie *reele* getallen moeten worden ingetikt.
  de rest van de opdracht geeft aan dat de waarden moeten worden opgeslagen in de variablen a,b,c.

- Merk op dat de namen van de variabelen in de scanf-opdracht elk worden voorafgegaan door een &. Dit heeft te maken met het begrip *pointer*, waar we later uitgebreid op zullen terugkomen.

- De scanf-opdracht wordt voorafgegaan door een printf-opdracht waarin wordt uitgelegd wat er ingetikt moet worden. Dit is niet verplicht, maar wel handig.

- U kunt de getallen *op een regel* intikken, van elkaar gescheiden door spaties; afsluiten met de *return*-toets.

- Ook de functie 'scanf' staat op de bibliotheek 'stdio.h', die wordt geactiveerd via de *include*-opdracht.

1.10 Mathematische functies

Het ligt voor de hand dat we bij numerieke berekeningen regelmatig *mathematische functies* zullen gebruiken. Een aantal daarvan is in C beschikbaar op een programmabibliotheek met de naam 'math.h'. Deze bibliotheek wordt geactiveerd met een *include*-opdracht.
Het volgende programma berekent de wortel van het product van twee getallen die U via het toetsenbord intikt.

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

int main (void)
{
    float a,b ;
    float wortel ;

    printf(" geef de waarden twee positieve getallen \n ");
    scanf ("%f %f", &a, &b);
    printf(" de ingetikte waarden van a b zijn : %f %f\n",a,b);

    wortel = sqrt ( a*b)      ;

    printf(" de wortel uit hun product is : %f \n", wortel) ;
```

```
    return 0 ;

}
```

Voorbeelden van beschikbare mathematische functies zijn :

sqrt(x)	levert $\sqrt{(x)}$
log(x)	levert $\ln(x)$
exp(x)	levert $\exp(x)$
pow(x,y)	levert x^y
fabs(x)	levert $\|x\|$
sin(x)	levert $\sin(x)$
cos(x)	levert $\cos(x)$
etc.	

1.11 Losse opmerkingen

- *Blanco* regels mogen naar willekeur worden ingevoegd (verhoogt de leesbaarheid).
- Tekst die is ingesloten tussen $/*$ en $*/$ wordt beschouwd als *commentaar* (verhoogt de leesbaarheid).
- Het *inspringen* van tekst is verplicht (verhoogt de leesbaarheid)
- Het gebruik van *spaties* binnen een regel is op veel plaatsen toegestaan, en kan de leesbaarheid verhogen.

1.12 Opdrachten

1.12.1 Opdracht 1

Welke van de volgende namen van variabelen zijn *niet legaal* ?

```
naam      quotient    x_1        x-1
eind!     4impuls     4_impuls   x12
```

1.12.2 Opdracht 2

Welke van de volgende arithmetische expressies zijn *niet in orde* ?

```
a+b       -c            d+(-f)
a+-c      (a+b)(c+d)    4.*((a-d)-(b-c)
```

1.12.3 Opdracht 3

Schrijf en test een programma dat :

- de waarde van drie reele getallen a,b,c inleest via het toetsenbord, en deze getallen uitprint.
- de wortels berekent van de vergelijking $ax^2 + bx + c = 0$
- de waarde van beide wortels uitprint.
- pas dit toe op a=1. b=-1.5 c=0.5
  a=2. b=-2. c=0.5
  a=1. b=-1.5 c=1.5

  NB : in het laatste geval 'gaat er iets mis'. Waarom ?

2 Control statements

2.1 Het if-statement

Vaak is het nodig om in Uw programma een deel van de code afhankelijk te maken van een of andere *voorwaarde/conditie*, zoals in het volgende voorbeeld (zie file 'ex1.c' in de workspace):

```
/* dit programma berekent de absolute waarde van een (ingetikt) getal    */
/* de code staat in de directory 'labwindows' onder */
/* http://www.nikhef.nl/user/h73/ccursus.html */
/* de code zit ook in de voorbeeld workspace voor labwindows */

#include <stdio.h>

int main (void)
{
    float x ;

    printf(" geef de waarde van een reeel getal \n ");
    scanf ("%f", &x);
    printf(" de ingetikte waarde van x is : %f \n",x);

    if ( x < 0 )
    {
       x = -x ;
    }

    printf(" de absolute waarde is : %f \n", x) ;

    return 0 ;
```

```
}
```

Dit is een voorbeeld van een *1-takkige if* :

- Er is een *voorwaarde* , 'verwoord' in de *relationele expressie* $x < 0$
- Als aan deze voorwaarde is *voldaan* dan wordt de 'code' tussen { } uitgevoerd.
- Als *niet* aan de voorwaarde is voldaan, dan wordt de 'code' tussen { } *overgeslagen*.

2.1.1 Relationele en logische expressies/voorwaarden

- In een *relationele expressie* worden de waarden van twee (arithmetische) constanten of variabelen met elkaar *vergeleken* via een *operator*. Mogelijke operatoren zijn :

$<$	kleiner dan
$<=$	kleiner dan, of gelijk aan
$>$	groter dan
$>=$	groter dan, of gelijk aan
$==$	gelijk aan
!=	ongelijk

- Het is mogelijk *meerdere* relationele expressies *samen* te voegen door middel van *logische operatoren*, zoals in :

```
/*   logische AND ; alleen 'waar' als aan beide voorwaarden voldaan is */
     if ( x < 0 && y != 0)

/*   logische OR ; 'waar' zodra aan een van beide voorwaarden voldaan is */
     if ( x < 0 || y != 0)
```

2.1.2 De eentakkige if

Deze vorm van het if-statement is reeds besproken.

2.1.3 De tweetakkige if

De tweetakkige if ziet er als volgt uit :

```
/*     dit (stukje) programma slaat de absolute waarde van de variabele 'x'
       op in de variabele 'absx'     */

    if ( x < 0 )
    {
       absx = -x ;
    }
```

```
else
{
   absx = x ;
}
```

- Als aan de voorwaarde is *voldaan*, dan wordt de code tussen de daaropvolgende set { } uitgevoerd.
- Als *niet* aan de voorwaarde is *voldaan*, dan wordt de code tussen de set { } , volgend op *else* uitgevoerd.

2.1.4 De meertakkige if

De meertakkige if (in dit geval een drietakkige) ziet er als volgt uit :

```
/*     in dit (stukje) programma krijgt de variabele y een waarde 1, 2 of 3
       als de variabele x negatief, nul of positief is. */

   if ( x < 0 )
   {
      y = 1 ;
   }
   else if ( x == 0 )
   {
      y = 2 ;
   }
   else
   {
      y = 3 ;
   }
```

- De voorwaarden worden in volgorde afgelopen. Zodra er aan eentje is voldaan wordt de code tussen de bijbehorende set { } uitgevoerd, en wordt *het geheel van de if verlaten.*
- Wanneer aan geen enkele van de voorwaarden is voldaan, wordt de code tussen de set { } , *volgend op else* uitgevoerd.
- Het *aantal* voorwaarden kan onbeperkt worden uitgebreid, door toevoegen van extra else if ( ) , met bijbehorende code .

2.2 De for-loop

Om de behoefte aan zoiets als een for-loop duidelijk te maken, bekijken we het probleem van *numeriek berekenen van de waarde van een integraal.*

2.2.1 Numeriek integreren

- Probleemstelling (funktie van 1 variabele).
  Gegeven $f(x)$ op $(a \leq x \leq b)$
  Bereken
  $$\int_a^b f(x)dx$$

- Om een Numeriek $recept$ te vinden, gaan we als volgt te werk:

  – Verdeel interval (a, b) in N intervallen van gelijke lengte h en representeer $f(x)$ in de vorm van een **tabel** :

  $$f_i = f(x_i) \quad x_i = a + ih \quad (i = 0, 1, 2, ..., N)$$

  – Schrijf de integraal als :

  $$\int_a^b f(x)dx = \sum_{i=0}^{N-1} \int_{x_i}^{x_{i+1}} f(x)dx$$

- Trapezium-regel (lineaire benadering) :
  Benader $f(x)$ op het interval (x_i, x_{i+1}) door :

  $$f(x) = \frac{f_{i+1} + f_i}{2} + \mathcal{O}(h)$$

  dan volgt :

  $$\boxed{\int_a^b f(x)dx \simeq \frac{h}{2}(f_0 + 2f_1 + 2f_2..... + 2f_{N-1} + f_N) + \mathcal{O}(h^2)} \tag{2.1}$$

Het is duidelijk zeer $bewerkelijk$ als we dit recept $stap\ voor\ stap$ zouden moeten coderen.

2.2.2 Voorbeeld van een for-loop

In het volgende voorbeeld wordt $\int_0^1 x^4 dx$ berekend; het interval $(\ 0,\ 1\ )$ wordt verdeeld in 100 sub-intervallen.

Zie de file 'ex2.c':

```
/* de code staat in de directory 'labwindows' */
/* onder http://www.nikhef.nl/user/h73/ccursus.html */
/* de code zit ook in de voorbeeld workspace voor labwindows */

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

int main(void)
{
```

```
     float a =   0., b = 1. ;         /* integratie grenzen      */
     float h ;                        /* interval-breedte        */
     float xi    ;                    /* x op intervalgrens      */
     float valint ;                   /* waarde van de integraal */
     int  nstep = 100 ;               /* aantal intervallen      */
     int  i ;                         /* teller in de for-loop   */

/*      bereken de intervalbreedte                                */
     h  = ( b - a ) / nstep ;

/*      eerst even de eindpunten                                  */
     valint = pow (a, 4.) + pow (b, 4.) ;

/*      nu de overige punten ; HIER KOMT DE FOR-LOOP              */
     for(i=1; i<nstep; i=i+1)
     {
        xi     = a + i * h ;
        valint = valint + 2. * pow (xi, 4.) ;
     }

/*      tot slot de vermenigvuldiging met h/2                     */
     valint = h / 2. * valint ;
     printf(" de numerieke waarde van de integraal = %f", valint) ;

     return 0 ;
}
```

De 'code' tussen de set { } , volgend op

```
     for(i=1; i<nstep; i=i+1)
```

wordt (in principe) *een aantal keren* doorlopen.

- De eerste keer heeft de variabele 'i' een waarde '1', zoals gespecifeerd in 'i=1' in het for-statement.
- De volgende keer is de waarde van 'i' met '1' opgehoogd, zoals gespecifeerd in 'i=i+1' in het for-statement.
- Dit gaat door *zolang* voldaan is aan de *voorwaarde* 'i<nstep' , zoals geformuleerd in het for-statement. Zodra *niet meer* aan deze voorwaarde is voldaan wordt de for-loop *verlaten*

2.2.3 Arrays van variabelen

- In het voorbeeld in de vorige (sub)sectie gaan de 'tussenliggende' waarden van de intervalgrenzen 'xi' en de bijbehorende functiewaarden *verloren*. In sommige

situaties in dit *ongewenst*, bijv. wanneer U een *grafiek* van de funktie wilt tekenen. Je zou dan de waarden van $x_0\ x_1$ ..... x_N en de bijbehorende funktie- waarden willen *bewaren* in *tabellen*.

- C biedt de mogelijkheid om variabelen een *dimensie* mee te geven; dit moet gebeuren *tegelijk* met de *type-declaratie*, zoals in :

```
float x[100] ;
```

  - Voor de variabele 'x' worden 100 geheugenplaatsen gereserveerd.
  - De getallen die daarin staan worden *aangesproken* via x[0], x[1], ... x[99]
  - **Merk op** dat de elementen worden *geteld vanaf* '0'. Dus : als de array 'x' een lengte '100' heeft , dan *bestaat* x[100] *niet*.

- In het volgende programma wordt de *tabel* 'nummer' gevuld met de getallen 1 t/m 10 :

```
#include <stdio.h>

int main (void)
{
    int nummer [10] ;
    int i ;

    for ( i=0; i<10; i=i+1 )
    {
       nummer [i] = i+1 ;
       printf(" element %d van array nummer bevat %d \n", i, nummer[i]);
    }

    return 0 ;
}
```

2.3 De while-loop en de do-while-loop

Bij de for-loop wordt een deel van de code een aantal malen doorlopen. U moet dit aantal *van te voren* specificeren *op de eerste lijn* van de for-opdracht.
Er zijn situaties denkbaar waarin U een deel van de code een aantal malen wilt herhalen, totdat een bepaalde *conditie* is bereikt, echter *zonder* dat U *van te voren* kunt specificeren hoeveel 'stappen' daarvoor nodig zijn.

- Bij de **while**-loop wordt de *voorwaarde* getest *voordat* de code tussen de set { } wordt uitgevoerd.

- Bij de **do while**-loop wordt de *voorwaarde* getest *nadat* de code tussen de set { } is uitgevoerd. Deze code wordt dus altijd *minimaal* een keer uitgevoerd.

- Het volgende voorbeeld (in file 'ex4.c') Ilustreert het verschil :

```
/* de code staat in de directory 'labwindows' */
/* onder http://www.nikhef.nl/user/h73/ccursus.html */
/* de code zit ook in de voorbeeld workspace voor labwindows */

/*      het is een illustratie van het verschil
        tussen while en do while                                  */
#include <stdio.h>

int main (void)
{
   int i = 0 ;

   while ( i < 0 )
   {

/*      deze print zal NIET verschijnen         */
      printf(" de while lus; i = %d\n", i ) ;
   }

   do
   {
 /*     deze print zal EEN MAAL verschijnen     */
      printf(" de do while lus; i = %d\n", i ) ;
   } while (i < 0 ) ;

   return 0 ;
}
```

2.4 Grafische weergave van resultaat

- Een programma produceert *altijd* een resultaat. Soms zijn dat *een* of *enkele* getallen, waarvan de betekenis onmiddelijk duidelijk is.
- Vaak is het resultaat echter 'het verloop' van een of andere grootheid *als functie van* een andere grootheid. Het uitprinten van reeksen getallen (bijv. in tabelvorm) verschaft dan *weinig inzicht* ; veel beter is dan een *grafiek*.
- In Labwindows zijn meerdere grafische functies beschikbaar. Hier beperken we ons tot een algemeen y tegen x plot met de functie 'XYGraphPopup'
- Hierna volgt het voorbeeld programma 'ex5.c' :

```
/* de code staat in de directory 'labwindows' onder */
/* http://www.nikhef.nl/user/h73/ccursus.html */
```

```
/* de code zit ook in de voorbeeld workspace voor labwindows */

#include <userint.h> // deze include zorgt voor het plotting user interface.
#include <math.h>

int main(void)
{
     float x[100]               ; /* de set x-coordinaten             */
     float y[100]               ; /* de bijbehorende waarden van f(x) */
     float dx = 0.1             ; /* intervallengte op de x-as        */
     int i                      ; /* teller voor de for-loop          */
     char title[] = "voorbeeld" ; /* 'titel' van de grafiek           */
 int err; // als er een error tijdesn de plotting optreedt wordt err ongelijk 0

/*        vul de arrays x[] en y[]             */
     for (i=0; i<100; i=i+1 )
     {
         x[i] = dx * i;
         y[i] = 2. * x[i] * x[i] ;
     }

   err = XYGraphPopup (title, x, y, 100, VAL_FLOAT, VAL_FLOAT);

     return 0;
}
```

2.5 Opdrachten

2.5.1 Opdracht 4 : het if statement : $ax^2 + bx + c = 0$

Dit is een variant op opdracht 3. Schrijf en test een programma dat :

- de waarde van drie reele getallen a,b,c inleest via het toetsenbord, en deze getallen uitprint.
  de wortels berekent van de vergelijking $ax^2 + bx + c = 0$
  de waarde van beide wortels uitprint.
- *beveilig* het programma tegen crashes door in de volgende gevallen de 'standaard-code' *niet* uit te voeren, en het probleem te signaleren via een *geprinte* tekst
  - als de discriminant < 0 is
  - als $a = 0$

- pas dit toe op a=1. b=-1.5 c=0.5
  a=2. b=-2. c=0.5
  a=1. b=-1.5 c=1.5
  a=0. b=-1.5 c=0.5

2.5.2 Opdracht 5 : arrays en het plotpakket

- Ook deze opdracht is een *variant* op het voorbeeld-programma in de sectie over de *for-loop*, waarin $\int_0^1 x^4 dx$ langs numerieke weg wordt berekend via de *trapezium-regel*.
- De opdracht luidt : herschrijf het voorbeeld-programma **ex2.c** zodanig dat :
  - De waarden van de interval-grenzen x_i en de bijbehorende functiewaarden *worden opgeslagen in arrays*
  - De waarde van de integraal wordt berekend *vanuit deze arrays*.
  - De functie x^4 op het interval $(0, 1)$ *grafisch* wordt weergegeven.
- **Lever in** :
  - *een laserprint van uw programma-code*
  - *een laserprint van de grafiek.*

3 Gebruik van functies in C

3.1 Inleiding

We hebben in de *voorbeeld*-programmas al eens kennis gemaakt met het gebruik/de aanroep van *functies* .

- Sommige functies werden door de C-compiler *meegeleverd*, zoals :

```
printf(" geef de waarden van de getallen a,b,c\n");
scanf ("%f %f %f", &a, &b, &c);

wortel = sqrt ( a*b)      ;

valint = pow (a, 3.) + pow (b, 3.) ;
```

Aan deze voorbeelden kunt U een aantal dingen *zien* :

- **Alle** functie-aanroepen worden gevolgd door ( ) Dit is **noodzakelijk** om er voor te zorgen dat de compiler 'begrijpt' dat het om een functie-aanroep gaat.
- Tussen de ( ) staan de *parameters* waarmee de functie wordt aangeroepen. Dit aantal kan per functie *verschillen*. De parameterlijst kan zelfs *leeg* zijn.

- Alle functies *doen* (uiteraard) iets, maar er zijn belangrijke *verschillen* :
  - *printf* levert (in dit geval) alleen een geprinte tekst, maar u kunt er ook de *waarde* van variabelen mee uitprinten.
  - *scanf* leest (in dit geval) drie getallen in, en zet de waarden in de variabelen die U 'a, b, en c' hebt genoemd; de *volgorde* is relevant.
  - *sqrt* trekt de wortel uit het getal dat in de parameterlijst staat, en stelt het resultaat ter beschikking *via de functie-naam*, zodat U het direct in een expressie kunt gebruiken.
  - *pow* gebruikt de twee parameters voor 'machtsverheffen', en stelt het resultaat ter beschikking *via de functie-naam*. De *volgorde* van de parameters is relevant.
  - In de functie *gpl_data* worden de parameters *gebruikt* om een grafiek te tekenen. Hun waarde wordt *niet veranderd*. Ook wordt er *geen resultaat teruggeven*.
    Het *aantal* en de *volgorde* van de parameters is relevant :U moet zich houden aan 'wat de functie verwacht'.
    Ook het *type* van de parameters is relevant.
    * de eerste is een 'enkele' variabele van het type 'int'
    * de tweede en derde zijn 'arrays' van het type 'float'
    * de vierde is een 'character string'.

    Ook hier moet U zich houden aan wat de functie verwacht !
  - *gpl_show* vertoont de aangemaakte grafiek in een window. er is *geen parameter* nodig, er is *geen functieresultaat*.

3.2 Het nut van functies

- Bovenstaande voorbeelden tonen minstens een *nuttig* aspect van het gebruik van functies : er is *1 stuk C-code* waarmee U *herhaalde malen* een bepaalde taak kunt laten uitvoeren, toegepast op *wisselende* (sets) parameters.
- Een *tweede* belangrijk voordeel is dat U door het gebruik van functies Uw code *in kleine modules* kunt inrichten, ieder met een *eigen taak*. Dit bevordert de overzichtelijkheid en de hanteerbaarheid van uw code, en verkleint daarmee de kans op 'fouten'.

3.3 Functies zonder parameter en zonder return-waarde

Een voorbeeld-programma (file 'ex6.c') :

```
/* de code staat in de directory 'labwindows' */
/* onder http://www.nikhef.nl/user/h73/ccursus.html */
/* de code zit ook in de voorbeeld workspace voor labwindows */
```

```
#include <stdio.h>

void mijnprint (void) ;      /* prototype van de functie */

int main (void)
{
   mijnprint () ;            /* aanroep van de functie   */
   return 0 ;
}
void mijnprint (void)        /* inhoud van de functie    */
{
   printf(" mijnprint is aangeroepen\n") ;
   return ;
}
```

Toelichting :

- Wanneer U functies gebruikt in Uw programma, moet U *voordat* de 'echte' code begint al iets vertellen over die functies, via het zgn. *prototype*. In

  ```
  void mijnprint (void) ;      /* prototype van de functie */
  ```

  'kondigt U aan' dat U (verderop) een functie gaat maken

  - met de *naam* 'mijnprint'
  - met een *lege* parameter-lijst (de 'void' rechts)
  - *zonder return*-waarde (de 'void' links)

  Merk op dat de prototype declaratie wordt afgesloten met een *punt-komma*.

- De *aanroep* van de functie vindt plaats in

  ```
  mijnprint () ;            /* aanroep van de functie   */
  ```

  Merk op

  - dat deze aanroep een *lege* parameterlijst heeft
  - dat *geen* return-waarde wordt verwacht

  Dit *klopt* dus met het 'prototype.

- De code van de functie zelf *begint* met

  ```
  void mijnprint (void)        /* inhoud van de functie    */
  ```

daarin komt weer terug : de *naam*, de *lege* parameterlijst, en het feit dat de functie *geen return*-waarde heeft.
Merk op dat hier *geen* punt-komma in voorkomt !

- De *inhoud* van de functie wordt *ingesloten* tussen { }
- De functie wordt *afgesloten* met

```
return ;
```

3.4 Functies met parameter en zonder return-waarde

Een voorbeeld-programma (file 'ex7.c'):

```
/* de code staat in de directory 'labwindows' */
/* onder http://www.nikhef.nl/user/h73/ccursus.html */
/* de code zit ook in de voorbeeld workspace voor labwindows */

#include <stdio.h>

void mijnprint (int) ;      /* prototype van de functie */

int main (void)
{
   mijnprint (10) ;         /* aanroep van de functie   */
   return 0 ;
}
void mijnprint (int getal)  /* inhoud van de functie    */
{
   printf(" mijnprint is aangeroepen met %d\n",getal) ;
   return ;
}
```

Toelichting :

- In het *prototype*

```
void mijnprint (int) ;      /* prototype van de functie */
```

'kondigt U aan' dat U (verderop) een functie gaat maken

  - met de *naam* 'mijnprint'
  - met *een* parameter
    het *type* van de parameter is 'int'
    de parameter is een *enkele* variabele (geen array)
  - de functie heeft *geen* return-waarde

- De *aanroep* van de functie vindt plaats in

```
mijnprint (10) ;            /* aanroep van de functie   */
```

  Merk op

  - dat deze aanroep *een* parameter heeft, van het type zoals aangegeven in het *prototype*
  - dat *geen* return-waarde wordt verwacht

- De code van de functie zelf *begint* met

```
void mijnprint (int getal)        /* inhoud van de functie    */
```

  - daarin komt weer terug : de *naam*, en het feit dat de functie *geen return*-waarde heeft.
  - bovendien wordt aangegeven dat de functie *een* parameter heeft, van het *type* 'int', en dat het geen array is.
  - bovendien wordt aangegeven dat we *binnen* de functie, voor de parameter de *naam* 'getal' zullen gebruiken.

- De *inhoud* van de functie wordt *ingesloten* tussen { }
  Merk op dat daar de variabele met de naam 'getal' inderdaad wordt gebruikt.

- De functie wordt *afgesloten* met

```
return ;
```

3.4.1 Opdracht 6

Maak een programma die een functie fibonacci bevat: $1, 1, 2, 3, 5, 8, 13$..... De functie het als input parameter het aantal stappen $nmax$ dat je in de for-loop moet doorlopen. Print uiteindelijk de verhouding tussen het laatste en het voorlaatse getal. Dat zou de zogenaamde Gulden Snede moeten opleveren van $(1 + \sqrt{5})/2 = 1.618$.

3.5 Functies met parameter en met return-waarde

Een voorbeeld-programma (file 'ex8.c'):

```
/* de code staat in de directory 'labwindows' */
/* onder http://www.nikhef.nl/user/h73/ccursus.html */
/* de code zit ook in de voorbeeld workspace voor labwindows */

#include <stdio.h>

float kwadraat (float) ;     /* prototype van de functie */
```

```
int main (void)
{
   float x = -1.0, xkwad ;

   xkwad = kwadraat (x) ;    /* aanroep van de functie   */

   printf ("het kwadraat van %f = %f \n", x, xkwad) ;

   return 0 ;
}
float kwadraat (float getal)  /* inhoud van de functie    */
{
   float x2 ;

   x2 = getal * getal ;

   return x2 ;
}
```

Toelichting :

- In het *prototype*

  ```
  float kwadraat (float) ;     /* prototype van de functie */
  ```

  'kondigt U aan' dat U (verderop) een functie gaat maken

  - Met de *naam* 'kwadraat'
  - met *een* parameter
    het *type* van de parameter is 'float'
    de parameter is een *enkele* variabele (geen array)
  - de functie heeft een return-waarde, en wel van het *type float*.

- De *aanroep* van de functie vindt plaats in

  ```
  xkwad = kwadraat (x) ;    /* aanroep van de functie   */
  ```

  Merk op

  - dat deze aanroep *een* parameter heeft, van het type zoals aangegeven in het *prototype*
  - dat een return-waarde wordt verwacht (die vervolgens wordt toegekend aan de variabele 'xkwad')

- De code van de functie zelf *begint* met

```
float kwadraat (float getal)  /* inhoud van de functie    */
```

  - daarin komt weer terug : de *naam*, en het feit dat de functie een *return*-waarde heeft van het *type* 'float'.
  - bovendien wordt aangegeven dat de functie *een* parameter heeft, van het *type* 'float', en dat het geen array is.
  - bovendien wordt aangegeven dat we *binnen* de functie, voor de parameter de *naam* 'getal' zullen gebruiken.

- De *inhoud* van de functie wordt *ingesloten* tussen { }
  Merk op dat daar de variabele met de naam 'getal' inderdaad wordt gebruikt.

- de opdracht

  ```
  return x2 ;
  ```

  zorgt er voor dat de waarde van 'x2' als *retourwaarde* van de functie wordt 'teruggeven'.

3.6 Array als parameter

Het is ook mogelijk om een *array* als parameter in een functie te gebruiken, zoals in het volgende voorbeeld (file 'ex9.c')

```
/* de code staat in de directory 'labwindows' */
/* onder http://www.nikhef.nl/user/h73/ccursus.html */
/* de code zit ook in de voorbeeld workspace voor labwindows */

#include <stdio.h>

int mijnsom( int [], int) ;         /* prototype van de functie */

int main (void)
{
   int a[10] = {0,1,2,3,4,5,6,7,8,9} ;
   int som ;

   som = mijnsom (a, 10) ;             /* aanroep van de functie */
   printf(" de som is %d\n", som) ;

   return 0 ;
}
int mijnsom (int reeks[], int lengte )  /* inhoud van de functie */
{
   int i, telop = 0 ;
```

```
   for (i=0; i<lengte; i=i+1)
   {
      telop = telop + reeks [i] ;
   }

   return telop ;                          /* resultaat van de functie */
}
```

Toelichting :

- In het *prototype*

  ```
  int mijnsom( int [], int) ;          /* prototype van de functie */
  ```

  'kondigt U aan' dat U (verderop) een functie gaat maken

  - met de *naam* 'mijnsom'
  - met *twee* parameters
    - het *type* van de eerste parameter is 'int' en het is een array (aangegeven door de [ ])
    - het *type* van de tweede parameter is 'int' en het is een 'enkelvoudige' variabele (geen array).
  - de functie heeft **een enkele** return-waarde van het *type* int

- De *aanroep* van de functie vindt plaats in

  ```
  som = mijnsom (a, 10) ;              /* aanroep van de functie */
  ```

  Merk op

  - dat deze aanroep *twee* parameters heeft, van het type zoals aangegeven in het *prototype*
  - dat de parameter 'a' **niet** wordt gevolgd door [ ]
  - dat een return-waarde wordt verwacht

- De code van de functie zelf *begint* met

  ```
  int mijnsom (int reeks[], int lengte )  /* inhoud van de functie */
  ```

  Het enige *verschil* met het prototype is (weer) dat de beide parameters nu ook een *naam* hebben gekregen, zodat ze 'binnen' de functie kunnen worden gebruikt.

- De *inhoud* van de functie wordt *ingesloten* tussen { }
  Merk op dat daar de variabelen met de naam 'reeks' en 'lengte' inderdaad worden gebruikt.

- de opdracht

  ```
  return telop ;
  ```

  zorgt er voor dat de waarde van 'telop' als *retourwaarde* van de functie wordt 'teruggeven'.

3.7 Functie resultaten in een array-parameter

Het kan voorkomen dat het *resultaat* van de berekeningen in een functie niet *een enkel getal* is, dat U via de functie-aanroep kun teruggeven, maar een *reeks van getallen in een array*.
In dat geval kunt U het resultaat 'teruggeven' *via de parameter-lijst*, zoals in het voorbeeld hierna (file 'ex10.c').

```
/* de code staat in de directory 'labwindows' */
/* onder http://www.nikhef.nl/user/h73/ccursus.html */
/* de code zit ook in de voorbeeld workspace voor labwindows */

#include <stdio.h>

void mijncop( float [], float [], int) ; /* prototype van de functie */

int main (void)
{
   float a[10] = {0.,1.,2.,3.,4.,5.,6.,7.,8.,9.} ;
   float b[10] ;
   int i ;

   mijncop ( a, b, 10 ) ;              /* aanroep van de functie */
   for(i=0; i<10; i=i+1)
   {
      printf(" i=%2d  a[i]=%5f  b[i]=%5f\n", i, a[i], b[i] ) ;
   }

   return 0 ;
}
void mijncop (float orig[], float copie[], int l)  /* inhoud van de functie */
{
   int i ;

   for (i=0; i<l; i=i+1)
   {
      copie[i] = orig[i] ;
   }
```

```
    return ;
}
```

Toelichting :

- In het *prototype*

  ```
  void mijncop( float [], float [], int) ; /* prototype van de functie */
  ```

  'kondigt U aan' dat U (verderop) een functie gaat maken

  - met de *naam* 'mijncop'
  - met *drie* parameters
    - het *type* van de eerste twee parameters is 'float' en het zijn arrays (aangegeven door de [ ])
    - het *type* van de derde parameter is 'int' en het is een 'enkelvoudige' variabele (geen array).
  - de functie heeft **geen** *return-waarde* via de functie-naam.

- De *aanroep* van de functie vindt plaats in

  ```
  mijncop ( a, b, 10 ) ;              /* aanroep van de functie */
  ```

  Merk op

  - dat deze aanroep *drie* parameters heeft, van het type zoals aangegeven in het *prototype*
  - dat de parameters 'a' en 'b' **niet** wordt gevolgd door [ ]
  - dat geen return-waarde wordt verwacht

- De code van de functie zelf *begint* met

  ```
  void mijncop (float orig[], float copie[], int l)  /* inhoud van de functie */
  ```

  Het enige *verschil* met het prototype is (weer) dat de parameters nu ook een *naam* hebben gekregen, zodat ze 'binnen' de functie kunnen worden gebruikt.

- De *inhoud* van de functie wordt *ingesloten* tussen { }

- de opdracht

  ```
  return ;
  ```

  sluit de functie af, op de manier zoals 'hoort' bij een void function.

3.8 Functieresultaten in een (enkelvoudige) parameter

Deze sectie over pointers en adressen van variabelen is uiterst belangrijk.

- U zou misschien verwachten dat het *teuggeven* van een functie- resultaat via een *enkelvoudige* parameter (geen array) op eenvoudige wijze kan worden 'afgeleid' uit het voorbeeld in de vorige sectie, nl. door het weglaten van de [ ] . Dit is echter **niet** het geval.

- De reden daarvan is dat de compiler in het geval van een enkelvoudige parameter, bij aanroep van de functie, de *waarde* van de parameter 'doorgeeft' , en *niet* het *adres* van de lokatie in het geheugen, waar de waarde van de parameter wordt bewaard.
  Zolang in de functie *alleen de waarde* wordt gebruikt, gaat het goed. Maar een berekende waarde *teruggeven* **kan niet**, althans niet op deze manier.

- Teruggeven van een berekende waarde *kan wel* op de manier, zoals aangegeven in het volgende *voorbeeld*. Daarbij wordt gebruik gemaakt van *pointers*, die vanuit de *waarde* van een parameter *verwijzen* naar het *adres* van zijn locatie in het geheugen.

Zie file 'ex13.c'.

```
/* de code staat in de directory 'labwindows' */
/* onder http://www.nikhef.nl/user/h73/ccursus.html */
/* de code zit ook in de voorbeeld workspace voor labwindows */

#include <stdio.h>

void mijncop( float, float *) ; /* prototype van de functie  */
                                /* let op * bij 2e parameter */
int main (void)
{
   float a = 10.;
   float b =  0.;

   mijncop ( a, &b) ;            /* aanroep van de functie    */
                                 /* let op & bij 2e parameter */

   printf(" a=%5f  b=%5f\n", a, b) ;

   return 0 ;
}
void mijncop (float orig, float *copie)  /* inhoud van de functie */
                                      /* let op * bij 2e parameter */
{
   *copie = orig ;                    /* let op *copie             */
```

```
   return ;
}
```

Merk op :

- Bij de $aanroep$ van de functie, in

  ```
  mijncop ( a, &b) ;            /* aanroep van de functie    */
  ```

  zorgt de '&b' er voor dat het $adres$ van de parameter wordt doorgegeven, in plaats van de waarde.

- Maar de functie moet dit natuurlijk ook 'weten', zowel via het 'prototype', als via de functie-'aanhef'. Het * bij de tweede parameter 'zorgt' daarvoor, in :

  ```
  void mijncop( float, float *) ; /* prototype van de functie  */
  void mijncop (float orig, float *copie)  /* inhoud van de functie */
  ```

- Dit * bij de parameter-naam moet ook gebruikt worden, als U een waarde in de parameter $invult$, zoals in

  ```
  *copie = orig ;                    /* let op *copie             */
  ```

Denk erom: je mag nooit iets lezen/schrijven uit/naar een pointer die niet geinitialiseerd is, dus:

```
float * aap;
*aap=0;
```

Dit is een 'dodelijk' statement om de pointer aap nog nergens naar wijst, terwijl je op deze plek een nul probeert te schrijven. Wat wel kan is:

```
float * aap;
float noot;
aap=&noot;  // aap bevat nu adres van noot
*aap=0;     // op het adres van noot wordt nu een nul geschreven.
```

De float noot bevat nu de waarde 0. Immers, eerst is de pointer aap op het adres van noot gezet. En daarna is op die locatie een nul geschreven. Het equivalent van dit omslachtig stukje code is dus gewoon: 'float noot=0;'. Niet meer en niet minder.

3.8.1 Opdracht 7

Maak nu een programma dat een functie bevat die de ABC formule bevat om een tweedegraads vergelijking op te lossen: $ax^2 + bx + c = 0$. De vijf parameters van de functie input van de functie zijn a,b,c en de output ans1 en ans2. Ook moet de functie integer waarde teruggeven die nul is als alles goed is gegaan en een andere waarde als er ene probleem is opgetreden, waardoor ans1 en ans2 niet berekend zijn. Dus iets als: int ABC(float a, float b, float c, float *ans1, float* ans2)

Print ans1 en ans2 in het hoofdprogramma (na aanroep van de functie ABC) en controleer of de functie doet wat er er wordt verwacht.

3.9 Communicatie via globale variabelen

Communicatie met globale variabelen is af te raden, omdat het in grote programma's tot problemen kan leiden. Voor kleine programma's is het echter wel de snelste en gemakkelijkste methode, maar toch is het niet aan te raden van globale variabelen gebruik te maken. In de voorgaande secties werd de *communicatie* tussen hoofdprogramma en functies 'geregeld' via de *parameter-lijst* en eventueel de *return-waarde* van de functie. Een *andere manier* van communicatie is via *globale variabelen*.

3.9.1 Lokale variabelen

- We hebben al eerder gezien dat variabelen worden gedefinieerd via een *type declaratie* ; daarbij wordt ook aangegeven of het een *enkelvoudige* variabele is, of een *array*.
- In al onze voorbeelden (tot nu toe) vond de type-declaratie plaats *binnen* het hoofdprogramma of *binnen* de functies, dwz *binnen* de bijbehorende set { }
- Het gevolg daarvan is dat de variabelen **alleen** 'betekenis' hebben *binnen* het hoofdprogramma of *binnen* de functie, waarin ze worden gedeclareerd, *zelfs als je dezelfde 'namen'* gebruikt.
- Een voorbeeld daarvan is het volgende programma (file 'ex11.c').

```
/* de code staat in de directory 'labwindows' */
/* onder http://www.nikhef.nl/user/h73/ccursus.html */
/* de code zit ook in de voorbeeld workspace voor labwindows */

#include <stdio.h>

void zomaar() ;                    /* prototype van de functie */

int main (void)
{
   int iks ;

   iks = 0   ;
   zomaar () ;                     /* aanroep van de functie */
   printf(" de waarde van iks in het main program = %d\n", iks ) ;

   return 0 ;
}
void zomaar (void)                 /* inhoud van de functie */
{
   int iks ;

   iks = 1 ;
```

```
   printf(" de waarde van iks in de functie = %d\n", iks ) ;

   return ;
}
```

Als U dit programma executeert, krijgt U de volgende printout :

```
 de waarde van iks in de functie = 1
 de waarde van iks in het main program = 0
```

Blijkbaar is de variable 'iks' in de functie *een andere* als in het main program.
In feite krijgen ze van de compiler gewoon *verschillende locaties* in het 'geheugen' toegekend.

3.9.2 Globale variabelen

De situatie wordt *heel anders* wanneer U de type declaratie *niet binnen* de { } van het main program of een functie doet, maar *daarbuiten*, en wel 'ergens aan het begin', (voordat de variabele wordt 'gebruikt'), zoals in het volgende voorbeeld (file 'ex12.c')

```
/* de code staat in de directory 'labwindows' */
/* onder http://www.nikhef.nl/user/h73/ccursus.html */
/* de code zit ook in de voorbeeld workspace voor labwindows */

#include <stdio.h>

int iks ;                          /* GLOBALE variabele        */

void zomaar() ;                    /* prototype van de functie */

int main (void)
{                                  /* iks wordt NIET (opnieuw) gedeclareerd */
   iks = 0   ;
   zomaar () ;                     /* aanroep van de functie */
   printf(" de waarde van iks in het main program = %d\n", iks ) ;

   return 0 ;
}
void zomaar (void)                 /* inhoud van de functie */
{                                  /* iks wordt NIET (opnieuw) gedeclareerd */
   iks = 1 ;
   printf(" de waarde van iks in de functie = %d\n", iks ) ;

   return ;
}
```

Als U dit programma executeert, krijgt U de volgende printout :

```
de waarde van iks in de functie = 1
de waarde van iks in het main program = 1
```

Blijkbaar is de variable 'iks' in de functie nu *dezelfde* als in het main program.
In feite krijgt een globale variabele van de compiler *een enkele locatie* in het 'geheugen' toegekend; zowel het main program als de functies krijgen *toegang* tot deze locatie.

Enige *voorzichtigheid* is geboden :

- Als U een variabele op deze manier 'globaal' maakt, mag U hem in de functie *niet nog eens* declareren (anders gaat het alsnog mis).
- De globale variable is (in zekere zin) *onbeschermd* : U kunt hem 'vanuit elke plek' *overschrijven*. Maar : zolang Uw programma niet al te groot is, is de kans hierop klein (als U goed oplet), en kan het gebruik van globale variabelen een handig *alternatief* zijn voor communicatie tussen de verschillende delen van Uw programma.
- Dit laatste is vooral handig als U in de functie *meerdere* getallen wilt laten berekenen/teruggeven, die *niet* samen *in een array* staan.
  - 'Teruggeven' van deze getallen via de functienaam kan niet (het zijn er meer dan een).
  - 'Teruggeven' via de parameterlijst *kan wel*, maar dan moet U *pointers* gebruiken.

4 Nog enkele 'handige zaken'

4.1 Gebruik van #define

- We pakken een 'oud' voorbeeld, waarin een array met lengte *10* wordt gevuld met de getallen 1 t/m *10* :

```
#include <stdio.h>
int main (void)
{
    int nummer [10] ;
    int i ;
    for ( i=0; i<10; i=i+1 )
    {
       nummer [i] = i+1 ;
       printf(" element %d van array nummer bevat %d \n", i, nummer[i]);
    }
    return 0 ;
}
```

- Stel dat U dit programma wilt veranderen zodat een array met lengte *100* wordt gevuld met de getallen 1 t/m *100*, dan moet U het programma *op twee plaatsen* wijzigen. Als U een plaats *vergeet*, dan gaat het mis.
  In het volgende programma wordt dat ondervangen :

```
#include <stdio.h>

#define lengte 10                  /* hier wordt 'lengte' gedefinieerd */

int main (void)
{
    int nummer [lengte] ;          /* gebruik 'lengte'                 */
    int i ;

    for ( i=0; i<lengte; i=i+1 )   /* gebruik 'lengte'                 */
    {
       nummer [i] = i+1 ;
       printf(" element %d van array nummer bevat %d \n", i, nummer[i]);
    }

    return 0 ;
}
```

- Overigens moet U zich realiseren dat hier *niet* een variabele wordt gedefinieerd met de naam 'lengte' en waarde '10'. Wat er wel gebeurt is het volgende :
  de *preprocessor* die uw code behandelt voordat de echte compilatie begint, *vervangt* in Uw code overal het de *tekst* 'lengte' door de *tekst* '10' .

4.2 Het gebruik van de 'break'-opdracht

Soms is het handig/nodig om een loop (for of while) *tussentijds* af te breken. Het volgende programma *test* Uw vermogen om gehele getallen (kleiner dan 100) te kwadrateren. Het programma *stopt* zodra U een fout maakt (file 'ex15.c').

```
/* de code staat in de directory 'labwindows' */
/* onder http://www.nikhef.nl/user/h73/ccursus.html */
/* de code zit ook in de voorbeeld workspace voor labwindows */


#include <stdio.h>

int main (void)
{
    int getal, kwadraat ;

    for ( getal=0; getal<100; getal=getal + 1)
```

```
    {
       printf(" geef het kwadraat van %d \n ", getal) ;
       scanf ("%d", &kwadraat) ;

       if ( kwadraat != getal * getal )
       {
           printf (" resultaat is fout bij het kwadraat van %d \n", getal) ;
           break ; /* stop zodra er een fout is gemaakt                    */
       }
    }

    return 0 ;
}
```

- In de for-loop worden de getallen in opklimmende volgorde aan U aangeboden.
- In het if-statement wordt gekeken of Uw antwoord goed is.
- Zodra U een fout maakt wordt Uw prestatie geprint, en wordt de for-loop *verlaten*

4.2.1 Opdracht 8

Herschrijf je ABC formule van de vorige opdracht zodat er alleen maar pointers in de parameterlijst staan. Als dat gelukt is, kun je ook nog een pointer (int *) als return waarde programmeren. (Meer voorbeelden m.b.t. pointers enzo staan in het project 'pointersenzo'.)

Volgend jaar bij 'Numerieke Natuurkunde' bouwwen we voort op de kennis die je bij deze cursus hebt opgedaan en gaan we ons meer richten op natuurkundige problemen die we numeriek op gaan lossen.

5 LabWindows specifiek

In het laatste deel van deze cursus gaan we ons alvast verdiepen in LabWindows specifieke functies, zodat jullie bij het volgende praktikum goed beslagen ten ijs komen.

Een voorbeeld van LabWindows zijn we tegengekomen bij het plotten van een grafiek met de functie 'XYGraphPopup' in Sectie 2.4. Deze functie zit standaard in Labwindow en we zullen in deze sectie door de stappen gaan waarmee we deze functie hebben ingebracht in onze code.

- maak een nieuw project en hang daarin een nieuwe C file. Zie eventueel Sectie 1.3.1. Begin een main programma dat nog niets doet, zoiets als in hello.c.
- In je labwindows scherm zie je in het scherm linksonderin 'Library' staan die je kunt aanklikken en verder uitvouwen. (Dit kan ook via het menu 'Library'). Klik vervolgens op 'User Interface library' en ga verder naar 'PopUp panels$-$ $>$Graph PopUps' en dubbelklik op 'XYGraphPopUp'. Er verschijnt een grafisch userinterface. Vul hier naar eigen inzicht redelijk namen in, bijvoorbeeld:

  - Title-veld: 'mijn plotje' (willkeurige naam)
  - X-Array-veld: 'xwaarden' (de naam van de array die je zal gaan gebruiken voor de x waarden).
  - Y-Array-veld: 'ywaarden'
  - Number-of-Points-veld: de lengte van de array's die je gaat gebruiken.
  - X-Data-Type-veld: we werken meestal met float's, dus kies 'floating point'
  - Y-Data-Type-veld: idem.
  - Status-veld: dit is de eventuele return waarde van de functie, een integer met willekeuruge naam: 'err' ofzo. Die moet overigens nul zijn na aanroep, anders is er een fout opgetreden.

- Op de gewone menu-balk is een extra optie 'Code' ontstaan, klik daarop en kies 'Set Target File' en zet deze op jouw nieuwe file.

- Vervolgens ook in menu 'Code', kies 'Insert Function Call' en nu heb je de functie in je code beschikbaar!

- vul je X-array en Y-array maar eens en roep de functie aan.

En nu gaan we hetzelfde doen met interactief paneel, maar daarvoor gebruiken we de officiele Labwindows manual, Hoofdstuk 5 in

http://www.nikhef.nl/~h73/docu_labwindows.pdf

Volg de aanwijzingen en maak een mooi interactief panel.