CPROUC/2021-2022

Jesse op den Brouw

CPROUC

Herhaling week 1 t/m 7

DE HAAGSE HOGESCHOOL

C-programma

- Een C-programma bestaat uit *statements*.
- Een statement is een opdracht voor de computer.
- Deze statements worden door de C-compiler vertaald naar *instructies* die de processor kan uitvoeren.
- Dit vertalen wordt *compileren* genoemd.
- Een statement kan complex zijn.
- Een instructie is eenvoudig
  - Tel op
  - Trek af
  - Vergelijk
  - Spring

C-programma

- Een eenvoudig C-programma:

```c
#include <stdio.h>

int main(void) {
    int a;
    scanf("%d", &a);
    printf("a = %d\n", a);
    return 0;
}
```

- Het programma leest een geheel getal (integer) van het toetsenbord en drukt het af op het beeldscherm.

Variabelen

- Een variabele is een stukje geheugen waarop bewerkingen worden gedaan.
- Een variabele moet kenbaar gemaakt worden aan de compiler. Dit wordt *definitie* genoemd.

```
char c = 'a';
int a, b = 3;
float x, y = 2.3f;
double pi = 3.14159265359;
```

- Een variabele mag gelijk *geïnitialiseerd* worden.

Variabelen

- Variabelen kunnen gebruikt worden in berekeningen:

```
int a, b = 2, c = 3, d = 4;
a = b * c + d;
```

- Normale rekenregels zijn van toepassing:
  - `*` `/` `%` gaan voor op `+` `-`

- Met haakjes kan de volgorde veranderd worden.

Beslissen

- C kent drie beslisstatements

- If
- If else
- Switch

- De zes relationele operatoren:

  == != > < >= <=

If

- Met if kunnen statements worden uitgevoerd als de *voorwaarde* waar (true) is

```
if (a > b)
{
   /* voer de statements uit als a > b */
}
```

If else

- Met if else kunnen statements worden uitgevoerd als de *voorwaarde* waar (true) of niet waar (false) is

```
if (a > b)
{
   /* voer de statements uit als a > b */
}
else
{
   /* voer de statements uit als a <= b */
}
```

Switch

- Met switch kan een expressie (meestal een variabele) op meerdere mogelijkheden getest worden

```
switch (a) {
    case 1: b = 5; break;
    case 2: b = 8; break;
    default: b = 1; break;
}
```

- Break is nodig om het switch-statement te verlaten.

Logische verbindingen

- Met && (en) en || (of) kunnen logische verbindingen gemaakt worden:

```
if (a > 5 && a < 10) { … }
```

```
if (a < 0 || a > 20) { … }
```

- Mogen meerdere keren gebruikt worden. && gaat voor op ||

Herhalen

- C kent drie herhalingsstatements:

- For
- While
- Do while

- Herhaling wordt uitgevoerd zolang de voorwaarde waar (true) is.

For

- For wordt gebruikt als het aantal herhalingen bekend is:

```
for (initialisatie; voorwaarde; aanpassen) { … }
```

- Voorbeeld:

```
for (i = 1; i < 11; i = i + 1)  { … }
```

- In dit voorbeeld “loopt” i van 1 t/m 10.

While

- While wordt gebruikt als het aantal 0 of meer is:

```
while (voorwaarde) { … }
```

- Voorbeeld:

```
while (a > b)  { … }
```

- De herhaling wordt uitgevoerd zolang a > b.

Do while

- Do while wordt gebruikt als het aantal 1 of meer is:

```
do { … } while (voorwaarde);
```

- Voorbeeld:

```
do { … } while (a > b);
```

- De herhaling wordt uitgevoerd zolang a > b (minstens 1 keer).

Functies

- Functies zijn stukjes programma die elders in het programma kunnen worden *aangeroepen*.

```
returntype functienaam(parameter, parameters, …)
{
    body
}
```

- Parameters mogen ook weggelaten worden → gebruik `void`
- Functie kan ook niets teruggeven → gebruik `void`

Functies

- Parameters gedragen zich als lokale, geïnitialiseerde variabelen.

```
int mul(int a, int b) {
    int c;
    c = a * b;
    return c;
}
```

- Het return-statement zorgt ervoor dat wordt teruggekeerd naar het aanroeppunt.

Functies

- Een functie aanroepen:

```
int main(void) {
    int z;

    z = mul(2, 3);
    printf("%d %d\n", z, mul(4, 5));

    return 0;
}
```

Array

- Een array is een rij variabelen onder één noemer. Een array bestaat uit *elementen*. Elk element heeft een *elementnummer*.

```
int rij[5]
```

- Nu zijn beschikbaar:

```
rij[0] rij[1] rij[2] rij[3] rij[4]
```

- Elementnummer mag berekend worden (integer).

Array – initialisatie

- We kunnen de array als volgt initialiseren:

```
int rij[5];
rij[0] = 2;
rij[1] = 3;
rij[2] = 7;
rij[3] = 1;
rij[4] = 8;
```

- Maar sneller is directe initialisatie:

```
int rij[] = { 2, 3, 7, 1, 8 };
```

Array – som bepalen

- Met behulp van een for-herhaling kunnen we “langs de array lopen”.

```
int rij[5] = { 2, 3, 7, 1, 8 };
int i, som = 0;

for (i = 0; i < 5; i = i + 1)
{
    som = som + rij[i]; /* i is het elementnummer */
}
printf("De som is: %d\n", som);
```

Array – aantal elementen uitrekenen

- Het is mogelijk om het aantal elementen van een array tijdens *compile-time* te berekenen.

```
int rij[] = { 2, 3, 7, 1, 8 };

int nr_elem = sizeof rij / sizeof rij[0];
```

- `sizeof` geeft het aantal *bytes* van het argument. Hier wordt dus de arraygrootte in bytes gedeeld door de grootte van één element in bytes.

Array – functies

- Een array kan dienen als een argument/parameter van een functie.
- Een array kan *niet* als returnwaarde dienen.
- Dat komt door de wijze van argument/parameter-overdracht.
- Bij een array-argument wordt niet de hele array overgedragen, maar het *adres van het eerste element*.
- Dat is veel efficiënter dan het de hele array overdragen.
- Het gevolg is dat een functie niet “weet” hoe groot een meegegeven array is.
- Er is dus een extra argument/parameter nodig met het aantal elementen van de array.

Array – functie berekenen som

- Een functie voor het berekenen van de som van de elementen in een array.

```
int berekensom(int r[], int grootte) /* r[] -> array */
{
   int som = 0;
   for (i = 0; i < grootte; i = i + 1)
   {
      som = som + r[i]; /* i is het elementnummer */
   }
   return som;
}
```

Array – aanroepen functie

- We kunnen nu de functie aanroepen.

```
int rij[] = { 2, 3, 7, 1, 8 };

int nr_elem = sizeof rij / sizeof rij[0];
int som;

som = berekensom(rij, nr_elem);
```

- Gebruik naam van array als argument.

Strings

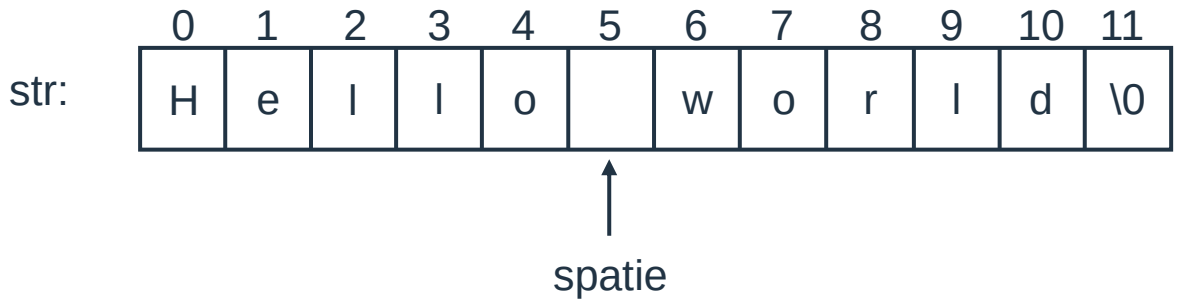
- Een string is een array van karakters afgesloten door een null-byte of null-karakter.
- Deze null-byte geeft het einde van de string aan.
- In een null-byte zijn alle bits 0.
- In C wordt dat geschreven als ‘\0’ (backslash-nul).
- Er is dus altijd één karakter extra nodig voor de ‘\0’.

Strings

- We kunnen een string bij definitie direct initialiseren, de null-byte wordt automatisch toegevoegd.

```
char str[] = "Hello world"; /* lengte automatisch bepaald */
```

- String in het geheugen:



DE HAAGSE HOGESCHOOL

Strings

- Een functie voor het bepalen van de lengte van een string is niet zo ingewikkeld:

```
int stringlength(char str[]) {
   int len;

   for (len = 0; str[len] != '\0'; len = len + 1);

   return len;
}
```

- Merk op dat de for-herhaling geen body heeft. De lengte wordt zonder body uitgerekend.

Strings

- De standard library kent een groot aantal string-functies. Voor gebruik moet het header-bestand `string.h` geladen worden.

```
strlen(str)          lengte string
strcpy(to, from)     kopiëren string
strcat(to, from)     from achter to plaatsen
strcmp(s1, s2)       vergelijk s1 met s2
```

- Als s1 en s2 identiek zijn, wordt 0 teruggegeven. Als s1 “kleiner” is dan s2, wordt een negatief getal teruggegeven. Als s1 “groter” is dan s2, wordt een positief getal teruggegeven.

Strings – array van strings afdrukken

- Voorbeeld van een array van strings afdrukken

```
char day[7][10] = { "zondag", "maandag", "dinsdag",
                    "woensdag", "donderdag", "vrijdag",
                    "zaterdag" };

int aantal = sizeof day / sizeof day[0]; /* bereken aantal strings */

for (int i = 0; i < aantal; i = i + 1)
{
   printf("%s\n", day[i]); /* day[i] → beginadres string */
}
```

Structure

- Een structure is een samengesteld datatype.
- De algemene gedaante van een structure is:

```
struct struct-identifier {
    datatype member1;
    datatype member2;
    ...
};
```

- De variabelen in de structure worden *members* genoemd.

Structure – gebruik members

- De members zijn te benaderen met de *member operator* .

```
struct punt {
   double x, y;
} p1, p2;

p1.x = 2.0; p2.y = 3.0;
p2.x = 7.5; p2.y = -0.5;

afstand = sqrt(pow(p1.x-p2.x, 2) + pow(p1.y-p2.y,2));
```

Structure – functie

- Een structure mag in zijn geheel dienen als argument/parameter.

```
void berekenafstand(struct punt pa, struct punt pb)
{
   return sqrt(pow(pa.x-pb.x, 2) + pow(pa.y-pb.y,2));
}
```

- De functie kan als volgt worden aangeroepen:

```
l = berekenafstand(p1, p2);
```

Structure – functie

- Een structure mag in zijn geheel dienen als returnwaarde:

```
struct punt lees_punt(void) {
    struct punt p;

    scanf("%lf %lf", &p.x, &p.y);

    return p;
}
```

Structure – array

- Array van structures met initialisatie:

```
struct punt p[50] =
{
   {  2.3, 7.3 },
   { -2.7, 6.9 },
   {  0.0, 4.9 }
};
```

- De overige structures worden met 0 geladen. Het aantal (50) mag weggelaten worden voor exacte allocatie.

Structure – afdrukken array

- Afdrukken van de array:

```
for (int i = 0; i < 3; i++)
{
   printf("%f %f\n", p[i].x, p[i].y);
}
```

- Uitrekenen aantal elementen:

```
int nr_elem = sizeof p / sizeof p[0];
```

Typedef

- Met `typedef` kan een nieuw datatype aangemaakt worden.
- Dit kan samen met de declaratie van de structure:

```
typedef struct punt {
   double x, y;
} punt_t;
```

- Nu kan de typedef gebruikt worden:

```
punt_t p1, p2;
```

Enum

- Met `enum` kan een enumeratie gemaakt worden. Dit wordt dan een nieuw datatype. Voorbeeld:

```
enum boolean { FALSE, TRUE };
```

- `FALSE` krijgt de waarde 0 en `TRUE` krijgt de waarde 1. Gebruik van het nieuwe datatype:

```
enum boolean x = TRUE;
…
if (x == TRUE) { … }
```

#include

- Met `#include` kan een header-bestand worden ingelezen (in principe elk bestand).
- Gebruik < en > voor systeem-header-bestanden:

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

- Gebruik " en " voor eigen gemaakte header-bestanden:

```
#include "studenten.h"
```

Macro's

- Met `#define` definieer je een macro:
- Simpele vorm:

```
#define AANTAL 10
```

- Nu kan je overal in het C-programma `AANTAL` gebruiken als *vervanging* van `10`.

```
int rij[AANTAL];
```

- Let op: `AANTAL` is geen variabele! De C-compiler ziet: `int rij[10];`

Macro's

- Een macro kan parameters hebben:

```
#define kwadraat(A) ((A) * (A))
```

- Gebruik:

```
int k = kwadraat(z + 1);
```

- De C-compiler ziet nu:

```
int k = ((z + 1) * (z + 1));
```

Conditionele compilatie

- Met behulp van `#if`, `#elif`, `#else` en `#endif` kan je bepaalde stukken C-programma compileren (of juist niet):

```
#if _MSC_VER == 1927
char compiler[] = "Visual Studio 2019 version 16.7";
#elif _MSC_VER == 1928
char compiler[] = "Visual Studio 2019 version 16.8 or 16.9";
#elif _MSC_VER = 1929
char compiler[] = "Visual Studio 2019 version 16.10 or 16.11";
#else
char compiler[] = "Visual Studio";
#endif
```

Conditionele compilatie

- Bij gebruik van debuggen (of niet)

```
int c; /* werk met variabele c */
…
#define DEBUG
…
#ifdef DEBUG
printf("DEBUG: c = %d\n", c);
#endif
```

- Alleen als DEBUG gedefinieerd is, wordt de `printf`-regel gecompileerd. De waarde van DEBUG is niet interessant

Conditionele inclusie

- Met behulp van `#ifndef` kan een header-bestand slechts één keer verwerkt worden

```
#ifndef _STUDENTEN_H
#define _STUDENTEN_H
   /* alle declaraties, typedefs, enums etc hier */
#endif
```

- De eerste keer dat dit bestand ingelezen wordt, is _STUDENTEN_H nog niet gedefinieerd en wordt de *body* verwerkt.
- Bij een volgende inlezing is _STUDENTEN_H wel gedefinieerd en wordt de body niet verwerkt.

Prioriteiten (eenvoudige versie)

Operator	Associativiteit
( )	Links naar rechts
! + - (unair) ++ --	Rechts naar links
* / %	Links naar rechts
+ -	Links naar rechts
< <= > >=	Links naar rechts
== !=	Links naar rechts
&&	Links naar rechts
||	Links naar rechts
= *= /= %= += -=	Rechts naar links

let’s change