VANDAAG

POINTERS

DYNAMISCH ALLOCATIE

MATRICES

STRINGS

FUNCTIES

WAT IS EEN POINTER?

EEN POINTER WIJST NAAR EEN LOKATIE IN HET GEHEUGEN VAN EEN COMPUTER

OF, ANDERS GEZEGD:

BEVAT HET ADRES VAN EEN VARIABELE

VOORBEELDEN

```
// declaratie (met type en *)
int *pointer_var;
int gewone_var;

// toekennen van adres
pointer_var = &gewone_var;
    // nu wijst pointer_var naar
    // het adres van gewone_var

// veranderen van waarde op adres
*pointer_var = gewone_var + 3;
    // de waarde van gewone_var is
    // nu met 3 opgehoogd
```

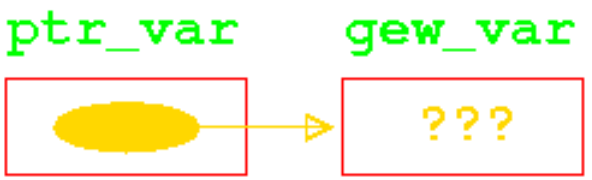
REKENEN MET POINTERS (1)

declaratie

```
int *ptr_var;
int gew_var;
```

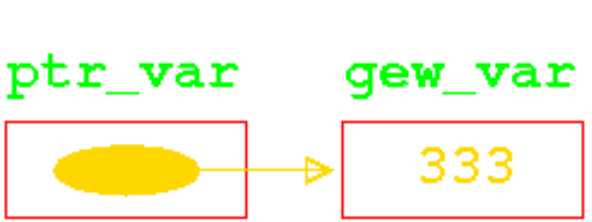
adres toekennen

```
ptr_var = &gew_var;
```



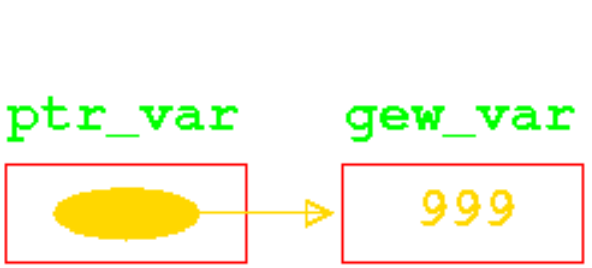
waarde toekennen

```
*ptr_var = 333;
```



waarde veranderen

```
gew_var *= 3;
```

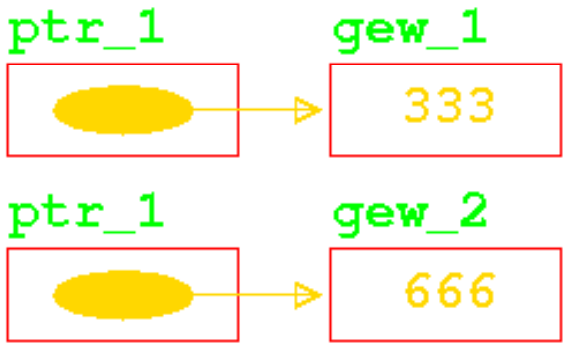


REKENEN MET POINTERS (2)

DECLARATIE EN INITIALISATIE

```
int gew_1 = 333;
int gew_2 = 666;
int *ptr_1;
int *ptr_2;

ptr_1 = &gew_1;
ptr_2 = &gew_2;
```



WAT IS HET VERSCHIL TUSSEN

`ptr_1 = ptr_2;` EN `*ptr_1 = *ptr_2;`

NULL-POINTER

EEN POINTER DIE DE GEHELE WAARDE 0 HEEFT, WIJST NAAR NIETS

I.P.V. HET GETAL 0 IS ER OOK DE CONSTANTE "NULL"

VOORBEELDEN

```
// declaraties
int *pointer_var_1 = 0;
int *pointer_var_2 = NULL;
int *pointer_var_3;

pointer_var_3 = NULL;

if (pointer_var_1 != NULL)
{
  *pointer_var_3 = *pointer_var_1;
}
```

DEREFERENCING

```
int var;  // declaratie
```

`&var` BETEKENT:

HET ADRES IN HET GEHEUGEN
VAN DE VARIABLE `var`

OFWEL:

DE POINTER NAAR VARIABELE `var`

OMGEKEERD:

DE INHOUD VAN EEN ADRES (REFERENCE)
WORDT VERKREGEN DOOR
"DEREFERENCING" D.M.V. EEN STERRETJE

```
int *ptr_naar_var;  // declaratie
*ptr_naar_var = 10; // toekenning
```

`ptr_naar_var`

IS EEN POINTERVARIABELE

POINTERS EN ARRAYS

```
#define maxDim 4

// declaratie en initialisatie
double arA[maxDim] =
   {1.0, 2.0, 3.0, 4.0};
double *ptr;
int    i;

// twee identieke toekenningen
ptr = &arA[0];
ptr = arA;

// druk alle array-elementen af
for (i=0; i<maxDim; i++)
{
   printf ("waarde arA[%d] = %lf\n",
            i, *(ptr+i));
}
```

DYNAMISCH GEHEUGEN

TIJDELIJK EXTRA GEHEUGEN GEBRUIKEN

```
// declaratie en allocatie
int *ptrI;
ptrI = (int *) malloc (20);
// (int *) voor juiste type pointer
// 20 bytes, niet 20 integers!
```

```
// declaratie en allocatie int
#define AANTAL 20
int *ptrI;
ptrI = (int *) malloc
            (AANTAL*sizeof(int));
```

GEHEUGEN VRIJGEVEN

TIJDELIJK EXTRA GEHEUGEN GEBRUIKEN

```
// declaratie en allocatie double
#define AANTAL 40
double *ptrD;
ptrD = (double *) malloc
            (AANTAL*sizeof(double));
```

EN EXTRA GEHEUGEN WEER VRIJGEVEN

```
// vrijgeven van gebruikte ruimte
free (ptrD);
// mits met malloc gereserveerd!
```

Pointer Fun with Binky

by Nick Parlante

This is document 104 in the Stanford CS Education Library — please see cslibrary.stanford.edu for this video, its associated documents, and other free educational materials.

Carpe Post Meridiem!

POINTER VIDEO

Binky Pointer Fun Video

DYNAMISCHE MATRICES

```
int **mat;          // pointer naar
                    //  array van arrays
int aKol, aRij;     // afmetingen
int kol, rij;       // loopvariabelen

// waarden voor aKol en aRij inlezen

// gewenste ruimte reserveren
mat = (int **) malloc
                    (aRij*sizeof(int));
for (rij=0; rij<aRij; rij++)
{ mat[rij] = (int *) malloc
                    (aKol*sizeof(int));
}

// matrix vullen
for (rij=0; rij<aRij; rij++)
{ for (kol=0; kol<aKol; kol++)
    mat[rij][kol] = rij * kol;
}
```

GEBRUIK VAN TEKST

```
char teken = 'q';    // enkel teken

#define LENGTE 4
 char *regel;          // string
// gewenste ruimte reserveren
regel = (char *) malloc
                (LENGTE*sizeof(char));

// een tekst wordt in C afgesloten
// het teken '\0', ofwel het getal 0
regel = "aap";
// geeft in het geheugen
```


```
// functie voor lengte van string
int lengte;
lengte = strlen (regel);
```

VOORBEELDEN

- Declaratie en gebruik van pointer variabelen
- Pointers en arrays
- Dynamisch alloceren van matrices, werken met matrices
- Pointers en strings

WAAROM FUNCTIES?

1) HERHALING VAN DEZELFDE CODE
   MET ANDERE PARAMETERS

2) VERDELING VAN PROGRAMMACODE IN
   FUNCTIONELE / LOGISCHE EENHEDEN

⇒ KORTERE PROGRAMMADELEN DIE

- OVERZICHTELIJKER ZIJN
- MINDER FOUTEN BEVATTEN

VERDEEL EN HEERS

(LEGIO VOORBEELDEN UIT GESCHIEDENIS)

ANALOGIE MET VERFIJNINGEN
VAN ALGORITMEN

FUNCTIES (1)

FUNCTIE VOERT DEELPROGRAMMA UIT

FUNCTIE KRIJGT INVOER VIA PARAMETERS VAN PARAMETERLIJST

FUNCTIE GEEFT RESULTAAT TERUG VIA FUNCTIEWAARDE

VOORBEELD

```
// functiedeclaratie
double cos (double angle)
{
  double value_of_cos;
  ... bereken cosinus ...
  return (value_of_cos);
}

...

double angle, // hoek in radialen
       cos_value;
...
angle = M_PI;
...
// functieaanroep
cos_value = cos (angle);
...
```

FUNCTIES (2)

OP DEZE WIJZE IS ER HEEL NETJES TE

PROGRAMMEREN !

MAAR . . .

SOMS IS DAT NIET ECHT HANDIG

BIJ HEEL VEEL INVOERPARAMETERS

BIJ VERSCHILLENDE TYPEN UITVOER

BIJ HEEL VEEL UITVOERVARIABELEN

DECLARATIE EN AANROEP

```
type functienaam (formele parameters)
{
   lokale declaraties;
   instructies;
   return (functiewaarde);
}

int main ()
{
   ...
   waardevanfunctie =
     functienaam (actuele parameters);
   ...
}


int kwadrateer (int grondtal)
{
   int uitkomst;
   uitkomst = grondtal * grondtal;
   return (uitkomst);
}

int main ()
{
   int getal=10, kwadraat;
   ...
   kwadraat = kwadrateer (getal);
   ...
}
```

FUNCTIES: PARAMETERLIJST

LEGE VARIABELEN

DIE BIJ AANROEP GEVULD WORDEN

"BY VALUE"	⟺	"BY REFERENCE"
GEBRUIK KOPIE VAN DE WAARDE (STANDAARD!)		GEBRUIK NAAM + INHOUD VAN ORIGINEEL
`printf ("%d", col);`		`scanf ("%d", &col);`

LOKAAL	⟺	GLOBAAL
BINNEN FUNCTIE		IN PROGRAMMA

GLOBALE VARIABELEN

```
// globale declaraties
static int G_a, G_b, G_c, G_d, G_e, ...

// functies
int  som1 (int ia, int ib)
void som2 (int ia, int ib, int *ic)
void som3 ()

// hoofdprogramma
int main ()
{
  int a, j, s;    // lokale declaraties

  // ken waarden toe aan globale variabelen
  G_a = 1;  G_b = 2;  G_c = 3;  G_d = 4;  G_e = 5;
  G_f = 6;  G_g = 7;  G_h = 8;  G_i = 9;  G_j = 10;

  // in- en uitvoer "zoals het hoort"
  a = G_a;
  j = G_j;
  printf ("Som   twee termen = %i\n", som1 (a, j));

  // uitvoer via parameterlijstvariabele
  som2 (2, 20, &s);
  printf ("Som   twee termen = %i\n", s);

  // bereken via globale variabelen en druk sommen af
  som3 ();
  printf ("Som oneven termen = %i\n", G_so);
  printf ("Som   even termen = %i\n", G_se);
  printf ("Som   alle termen = %i\n", G_sa);

  return (0);
}
```

VOORBEELDEN

- Type van functie int <> void
- Parameters van functie: by value <-> by reference
- Diverse manieren om waarden in en uit een functie te krijgen
- Functies: neveneffecten
- Hello World met input parameters

VOORBEELD ALGORITHME

Algoritme voor het bepalen van het nulpunt van een strikt stijgende functie tussen xL en xH (opg 9)

- nulpunt bij xL?
  ja, geef nulpunt terug, KLAAR
- nulpunt bij xH?
  ja, geef nulpunt terug, KLAAR
- functiewaarde verandert van teken tussen xL en xH?
  nee, geen nulpunt tussen xL en xH, KLAAR
- voor N stappen
  - bepaal midden mP = ½ (xL + xH)
  - nulpunt bij mP?
    ja, geef nulpunt terug, KLAAR
  - functiewaarde mP > 0?
    ja, stel bovengrens van interval bij: xH = mP
    anders, stel ondergrens van interval bij: xL = mP
- geef nulpunt van mp = ½ (xL + xH) terug, KLAAR