EE1400: Programmeren in C

BSc. EE, 1e jaar, 2012-2013, 3e college

Arjan van Genderen, Computer Engineering

4-12-2012

Hoorcollege 3

- Arrays, Pointers en Strings
- Bitwise operators
- Type definities
- De C preprocessor

Corresponderende stof in boek: Hoofdstuk 6 t/m 8

Arrays, Pointers en Strings

Arrays

Vaak homogene data in een programma:

```
int grade0, grade1, grade2;
```

kan efficienter met arrays:

```
int grade[3];   /* creates grade[0], grade[1], grade[2] */
```

arrays beginnen altijd bij 0 in C

Ander voorbeeld:

```
#define  N  100
int a[N];   /* space for a[0], a[1], ..., a[99] */

sum = 0;
for (i = 0; i < N; ++i)
   sum += a[i];
```

Index kan elke integer expressie zijn:

```
a[(i + 1)*2];
```

Wanneer index buiten range kunnen willekeurige fouten optreden:

```
printf ("%d\n", a[-1]);
  /* arbitrary value printed or runtime error message !! */
```

Initialisatie

Array declaratie met initialisatie:

```
float f[5] = {0.0, 1.0, 2,0, 3.0, 4.0};
```

equivalent aan:

```
float f[] = {0.0, 1.0, 2,0, 3.0, 4.0};
```

Multi-dimensionele Arrays

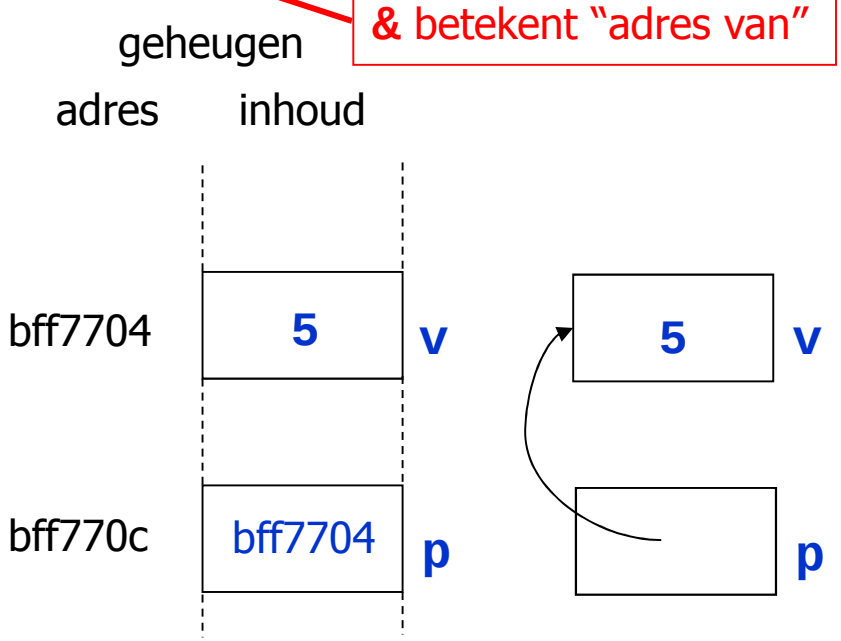
```
int a[2][3];

int a[2][3] = {1, 2, 3,
               4, 5, 6};

int a[][3] = {{1, 2, 3},
              {4, 5, 6}};

int a[2][2][3] = {{{1, 1, 0}, {2, 0, 0}},
                  {{3, 0, 0}, {4, 4, 0}}};
```

Pointers

Een pointer bevat het adres (de geheugenlocatie) van een variabele

```
int v;    /* variabele van het type int */
int *p;   /* pointer naar een variabele van type int */

v = 5;
p = &v;   /* p bevat nu het adres van v (p points to v) */
```

& betekent "adres van"



symbool * geeft bij gebruik in een statement de waarde aan van de variabele waar de pointer naar wijst:

```
int w = *p; /* oftewel w = 5; */
*p = 10;    /* oftewel v = 10; */
```

Meer over Pointers

```
#include <stdio.h>

int main (void)
{
   int i = 7, *p;

   p = &i;

   printf ("%s%d\n%s%p\n", "   Value of i: ", *p,
                           "Location of i: ",  p);
}
```

Output:

```
   Value of i: 7
Location of i: effffb24
```

Merk op: `*&i` is gelijk aan `i`

```
&3              /* illegal */
&(k + 99)       /* illegal */

int *p = 100;         /* illegal */
int *p = 0;           /* ok, pointing to nothing */
int *p = (int *)q;    /* ok, but be careful */
```

Pointers als functie parameters

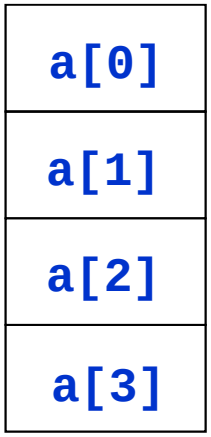
```
#include <stdio.h>

void swap(int *p, int *q)
{
   int   tmp;

   tmp = *p;    /* tmp = 3 */
   *p = *q;     /* i = 5 */
   *q = tmp;    /* j = 3 */
}

int main(void)
{
   int    i = 3, j = 5;

   swap(&i, &j);
   printf("%d  %d\n", i, j);        /* 5  3 is printed */
   return 0;
}
```

adressen van i en j worden doorgegeven, zodat in swap de inhoud van i en j verwisseld kan worden

Relatie Arrays en Pointers

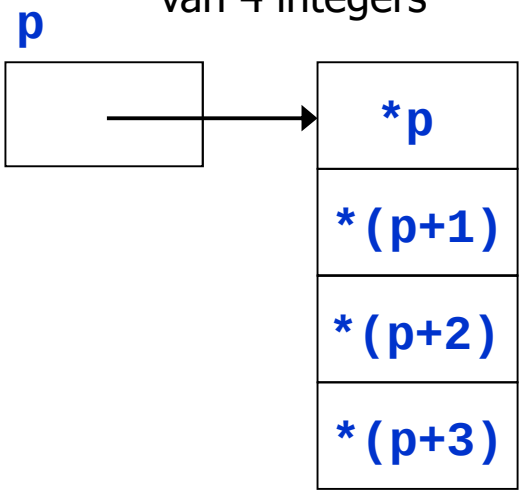
```
int a[4];
```

geheugen indeling voor a




```
int *p;
```

stel: p wijst naar begin van 4 integers



Mede daarom is verder gedefinieerd dat:

```
a[i]      is gelijk aan   *(a + i)
*(p + i)  is gelijk aan   p[i]
```

En:

```
p = a;      is gelijk aan   p = &a[0];
p = a + i;  is gelijk aan   p = &a[i];
```

Maar:

`a = p;` is illegaal

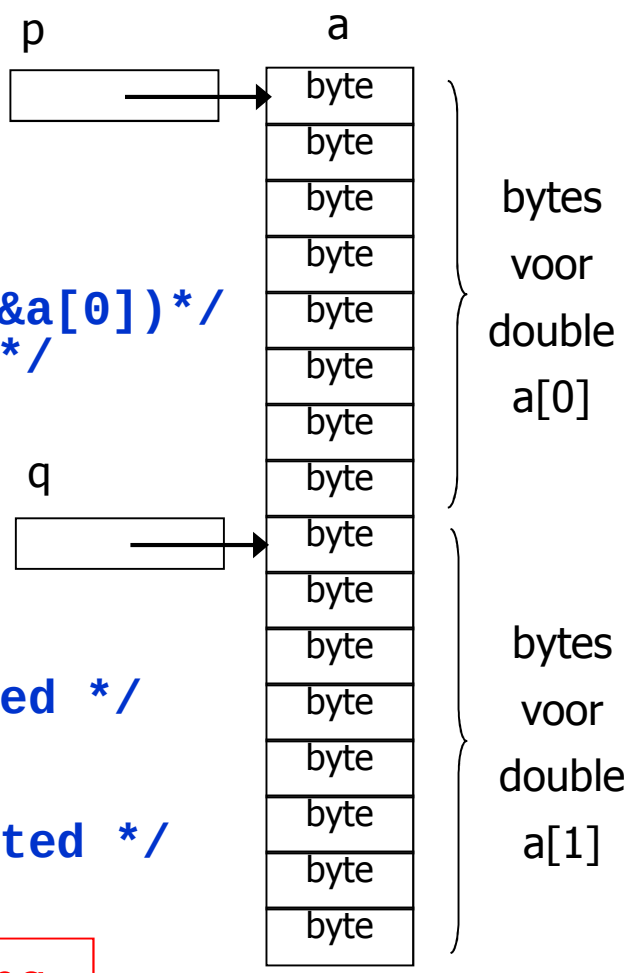
Pointer arithmetiek

```
double a[2], *p, *q;

p = a;   /* points to base of array a (&a[0])*/
q = p + 1;  /* equivalent to q = &a[1] */
```

compiler weet dat het verschil 1 double is

```
printf ("%d\n", q - p)    /* 1 is printed */

printf ("%d\n",
        (int)q - (int)p);  /* 8 is printed */
```

pointer *waarden* hebben betrekking op byte adressen: de volgende double staat 8 bytes verder



sizeof (double) = 8 !

Arrays als functie argument

```
double sum (double a[], int n)  /* n is the size of a[] */
{
   int     i;
   double  sum = 0.0;

   for (i = 0; i < n; ++i) {
      sum += a[i];
   }
   return sum;
}
```

- Het base adres van een array (&a[0]) wordt doorgegeven.
- De array elementen zelf (a[0], a[1], ... a[n-1]) worden niet gekopieerd.
- Array element binnen functie veranderd → ook buiten functie veranderd.

De volgende functie header is equivalent:

```
double sum (double *a, int n)  /* n is the size of a[] */
{
   ...
```

Voorbeeld: Bubble Sort

```
void swap (int *, int *);

void bubble(int a[], int n)      /* n is the size of a[] */
{
   int    i, j;

   for (i = 0; i < n - 1; ++i)
      for (j = n - 1; j > i; --j)
         if (a[j-1] > a[j])
            swap(&a[j-1], &a[j]);
}
```

zie slide 9

Aanroep bijvoorbeeld:

```
bubble (a, 8);
```

	a[0]	a[1]	a[2]	a[3]	a[4]	a[5]	a[6]	a[7]
start	7	3	66	3	-5	22	-77	2
i=0	-77	7	3	66	3	-5	22	2
i=1	-77	-5	7	3	66	3	2	22
i=2	-77	-5	2	7	3	66	3	22
i=3	-77	-5	2	3	7	3	66	22
i=4	-77	-5	2	3	3	7	22	66
i=5	-77	-5	2	3	3	7	22	66
i=6	-77	-5	2	3	3	7	22	66

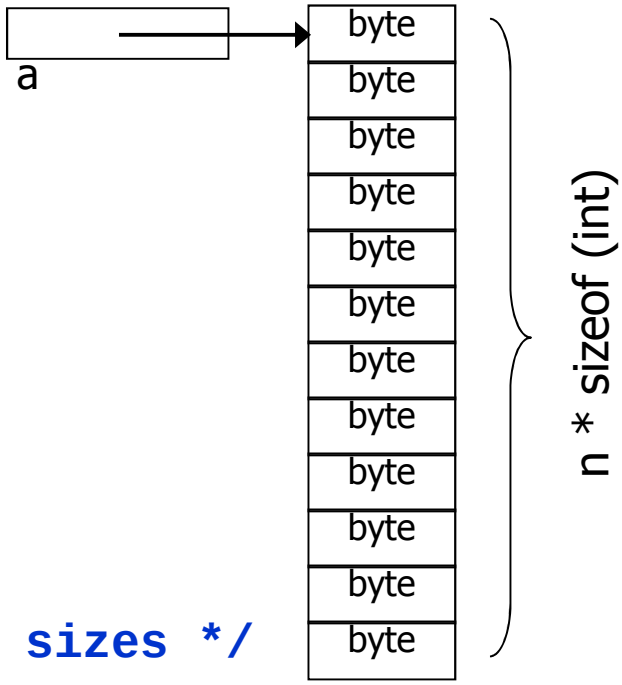
Dynamische geheugen allocatie

Geheugen allocatie van n integers aan pointer a:

```
#include <stdlib.h>
int main(void)
{
   int *a;
   int n;

   ...
   a = calloc (n, sizeof (int));
       /* all bits are set to 0 */
   a[i] = ...
```

Of:

```
   a = malloc (n * sizeof (int));
     /* unintialized; faster for large sizes */
```



Wanneer geen geheugen beschikbaar: NULL (0) wordt geretouneerd

Geheugen weer vrijgeven: `free (a);`

Strings

Strings zijn arrays van characters

```
char s[] = "abcdef";
char s[] = {'a', 'b', 'c', 'd', 'e', 'f', '\0'};
```

(equivalente declaratie)

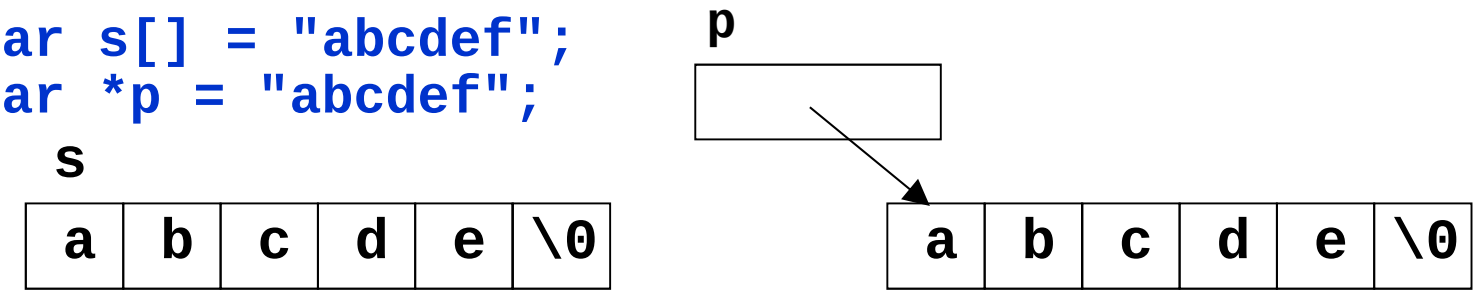
Pointers kunnen ook gebruikt worden:

```
char *p = "abc";
printf ("%s %s\n", p, p + 1);  /* abc bc is printed */
```

p wijst naar eerste karakter van de array: 'a'

Verschil tussen:

```
char s[] = "abcdef";
char *p = "abcdef";
```



Standard string functies

De standard library bevat allerlei functies voor string handling.

Bijvoorbeeld: strcmp (s1, s2) vergelijkt strings s1 en s2 lexicografisch en retouneert 0 indien gelijk, -1 wanneer s1 < s2, en 1 wanneer s1 > s2.

```
int strcmp (char *s1, char *s2);
{
   char c1, c2;

   while (*s1 != '\0' && *s1 == *s2) {
      s1++;
      s2++;
   }

   c1 = *s1;
   c2 = *s2;
   return ((c1 < c2) ? -1 : (c1 > c2));
}
```

ga door zolang geen einde string en zolang gelijk

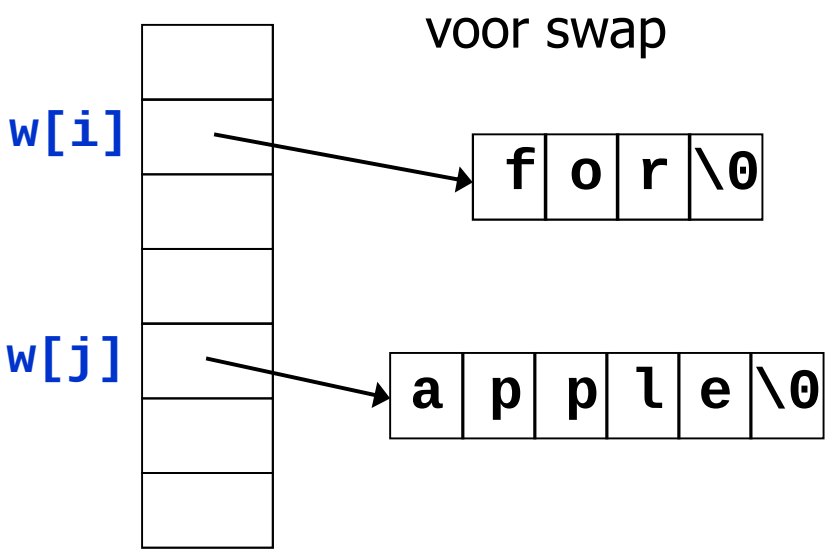
waarde -1 wanneer c1 < c2, anders resultaat van c1 > c2

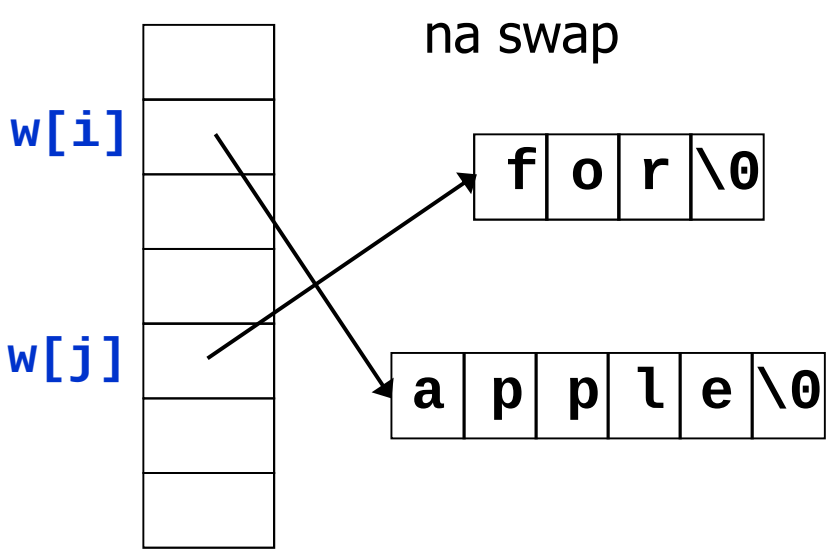
Arrays van pointers

```
void sort_words(char *w[], int n) /* n words to be sorted */
{
   int   i, j;

   for (i = 0; i < n; ++i)
      for (j = i + 1; j < n; ++j)
         if (strcmp(w[i], w[j]) > 0)
            swap(&w[i], &w[j]);
}
```

array van pointers naar string

char * versie van slide 9



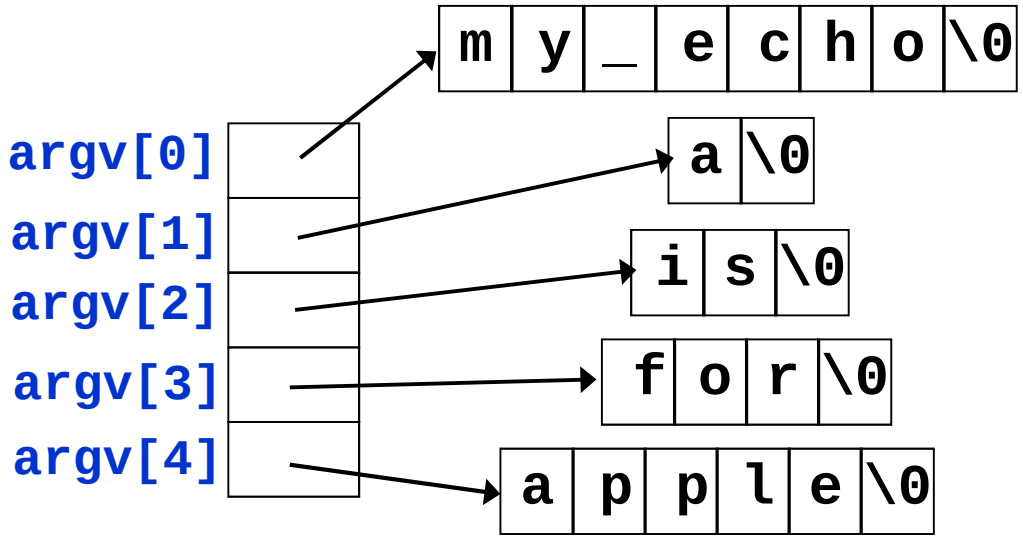


Programma argumenten

```
#include <stdio.h>

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

   printf ("argc = %d\n", argc);
   for (i = 0; i < argc; i++)
      printf ("argv[%d] = %s\n", i, argv[i]);
}
```

Run commando:
```
my_echo a is for apple
```

Output:
```
argc = 5
argv[0] = my_echo
argv[1] = a
argv[2] = is
argv[3] = for
argv[4] = apple
```



Bitwise operators

Bitwise operators

Stel

```
int a = 5;
int b = 6;
```

De binaire representaties van a en b zijn (bij 32 bits int):

```
00000000 00000000 00000000 00000101
00000000 00000000 00000000 00000110
```

Wanneer

```
int c = a & b;
```

dan zullen de bits van a en b bitwise ge-AND worden:

```
00000000 00000000 00000000 00000100
```

Oftewel c zal de decimale waarde 4 krijgen.

Bitwise binaire logische operaties

operaties op bits a_i en b_i

a_i	0	1	0	1	
b_i	0	0	1	1	
a_i & b_i	0	0	0	1	and
a_i ^ b_i	0	1	1	0	exclusive or
a_i | b_i	0	1	1	1	inclusive or

Wanneer a en b van het type int zijn:

expressie	representatie	waarde
a	**00000000 00000000 10000010 00110101**	**33333**
b	**00000000 00000000 11010000 00101111**	**53295**
a & b	**00000000 00000000 10000000 00100101**	**32805**
a ^ b	**00000000 00000000 01010010 00011010**	**21018**
a | b	**00000000 00000000 11010010 00111111**	**53823**

Shift operaties

Bits naar links schuiven:

```
int a = 12;  /* a = 00001100 */
a = a << 1;  /* a = 00011000 (24) */
```

Bits naar rechts schuiven:

```
int a = 16;  /* a = 00010000 */
a = a >> 3;  /* a = 00000010 (4) */
```

n bits naar links komt overeen met: vermenigvuldigen met 2^n

n bits naar rechts komt overeen met: delen door 2^n

Testen of het n^e bit van rechts in een variabele z de waarde 1 heeft:

```
if (z & (1 << (n-1))
```

Type definities

Nieuwe type definities

Nieuwe typen kunnen worden geintroduceerd met typedef:

```
typedef  char  uppercase;
uppercase u;


typedef  int   INCHES, FEET;
INCHES  length, width;


typedef  double scalar;
typedef  scalar vector[N];
vector y;
y[0] = 4.0;
```

Enumeratie typen

```
enum day {sun, mon, tue, wed, thu, fri, sat};
enum day d1 = sun, d2 = mon;

typedef  enum day  day;

day find_next_day(day d)
{
   day next_day;

   switch (d) {
      case sun:
         next_day = mon;
         break;
      case mon:
         next_day = tue;
         break;

      ...
   }
   return next_day;
}
```

De C preprocessor

Macros

geen spatie toegestaan

```
#define  SQ(x)  ((x) * (x))
```

`SQ(7 + w)` expandeert naar `((7 + w) * (7 + w))`

sneller dan een functie aanroep!

Minder handig is:

```
#define  SQ(x)  x * x
```

want `SQ(a + b)` expandeert naar `a + b * a + b`

Ook niet handig is:

```
#define  SQ(x)  (x) * (x)
```

want `4 / SQ(2)` expandeert naar `4 / (2) * (2)`

Andere voorbeelden:

```
#define  MIN(x, y)   (((x) < (y)) ? (x) : (y))

#define  MY_MALLOC(s, type, n)                          \
    if ((s = malloc(sizeof(type) * n)) == NULL) {       \
       printf ("Not enough memory\n");                  \
       exit (-1);                                       \
    }
```

Voorwaardelijke compilatie

```
#ifdef DEBUG
   printf("debug: a = %d\n", a);
#endif
```

```
#ifndef DICE_H_DEFINED   /* to prevent multiple inclusion */
#define DICE_H_DEFINED   /* of code in a header file */
void init_dice (void);
void throw_dice (int dice[], int reroll_flags[]);
#endif
```

```
#if INT_MAX > 100000
typedef  unsigned int    size_t;
#else
typedef  unsigned long   size_t;
#endif
```

Samenvatting

- Arrays, Pointers en Strings
- Bitwise operators
- Type definities
- De C preprocessor