Programming for Beginners

Lecture 4: introduction to arrays & pointers

Kai Lampka

Uppsala University

kai.lampka@it.uu.se

Arrays

- In C one may define a matrix of elements of a specific data type.
- Such matrices are denoted as multi-dimensional arrays or in case of a
- vector as array.
- We access the individual elements of the array by an index which is just another (counting) variable
- The index start with value 0 and range up to n-1 if n is the number of elements of the array
- to each element of the array one may apply mathematical operations suitable for the type of the element.

```
int n = 10;
int a[n];
for(int i = 0; i > n; i++) a[n] = i;
```

Syntax

- At first you give the type of the elements, in the example they are integers
- the data type is followed by the identifier of the array
- the identifier is followed by the number of elements you want to store in the array in []-brackets. This number is fixed throughout the life time of the array!
- One may also give a list of initial values, where the first value is assigned to position 0 in the array , etc.
- If there is no initial value given, the array element is initialized, i.e, it has a random value

```
int n = 10;
int a[n] = {3,5,7,11,13,17,19,23, 27, 31};

// initialization of some elements is also possible
int b[n] = {3,5,7};

// If all elements are initialized,
// the compiler figures out how many there are
int c[] = {3,5,7,11,13,17,19,23, 27, 31};
```

Arrays

```
#include<stdio.h>

int main(){
    int a[4];
    int i;

    for ( i = 0; i < 4; i++ )
        a[i] = 0;

    for ( i = 0; i < 4; i++ )
        printf("a[%d] = %d\n", i , a[i]);

    return 0;
}
```

Arrays

```
#include<stdio.h>

int main(){

    int a[10];
    int i;
    char in;

    for ( i = 0; i < 10; i++ ){
        printf (“Please give a number”);
        scanf(“%c”, &c);
        a[i] = atoi(&c);
    }
    for ( i = 0; i < 10; i++ )
        printf("a[%d] = %d\n", i , a[i]);

    return 0;
}
```

From Arrays to Pointer

Important remark:

- the elements of an array are stored sequential in the memory, one-by-another and neighboring.
- the identifier of the array in fact refers to the address of the first element
- This way one may actually compute the position (address) of each element and access the address directly.
- With the asterix (*) we tell the compiler that we want to access the actual contents stored at the specified address.

```
int n = 10;
int a[n] = {3,5,7,11,13,17,19,23, 27, 31};

for(int i = 0; i > n; i++)
 printf("Element %d of %d: %d\d",
          i, n, *(a+i);
```

Pointer

- for each built-in or self-defined data type one can allocate arrays
- the identifier of an array is actually a pointer to the starting address of the first element of the array.
- Instead of defining a whole array we may simply define pointers which point to some entry in the array including the first element.
    - When iterating over an array one may need to access different elements, pointers are an efficient way to do so
    - The size of an array may change over a programmes lifetime, with pointers one can easily organizing the re-allocation of arrays
- The empty pointer is defined NULL
- Operators to pointers:

    * : the asterix gives the contents of a pointer, e.g., in case of an array, it returns the respective element the pointer is pointing to.

    &: the ampersant returns the address of a pointer, i.e., the location there the pointer is actually stored

Pointer

```
int main () {
    int n = 10, max = 0, i;
    int a[n] = {3,5,7,11,13,17,19,23,27,31};

    for(i = 0; i < n; i++)
        printf("Element %d of %d: %d\d",i, n, *(a+i);

    for(i = 0; i < n; i++){
        if(max <= a[i]) continue;
        else max = a[i];
    }

    for(int *b = a; b < (a+n-1); b++){
        if(max <= (*b)) continue;
        else max = (*b);
    }
    return 0;
```

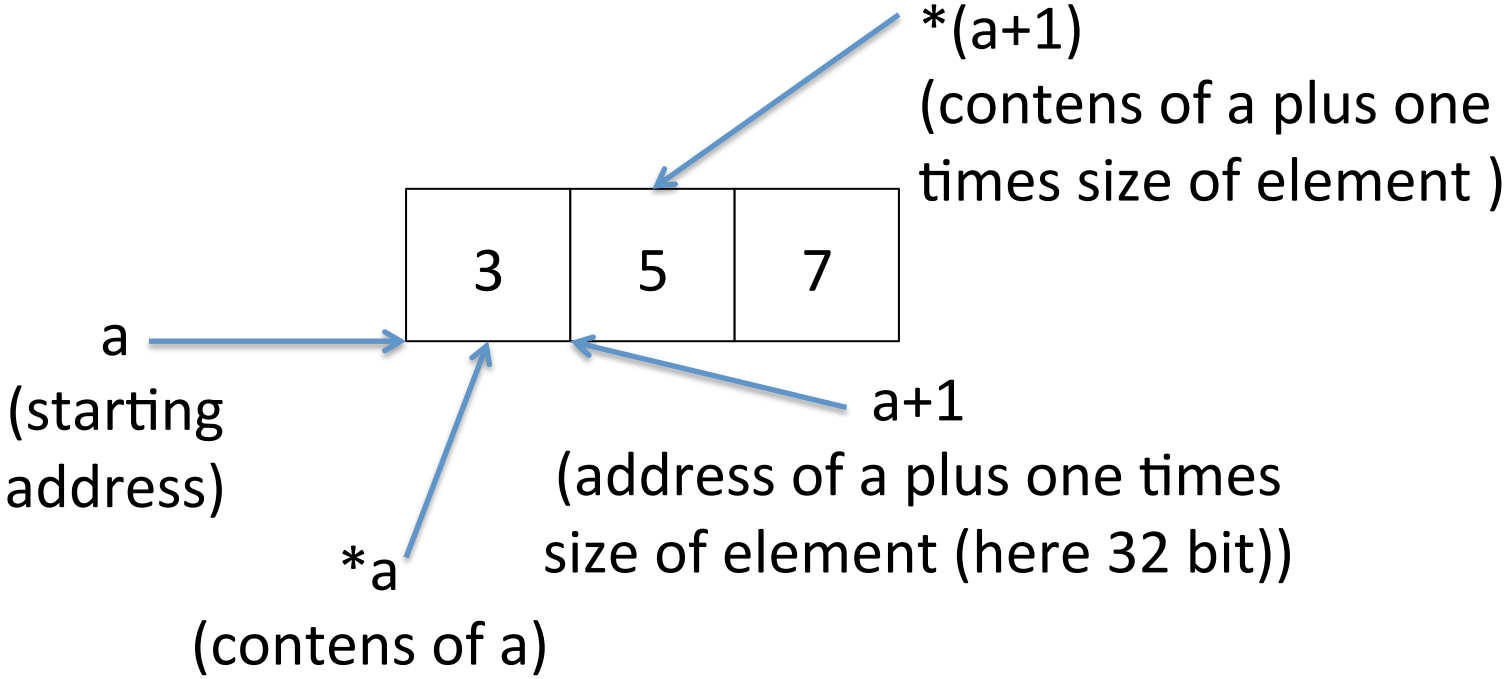
Pointer

✧ Operators to pointers:

* : contents of a pointer

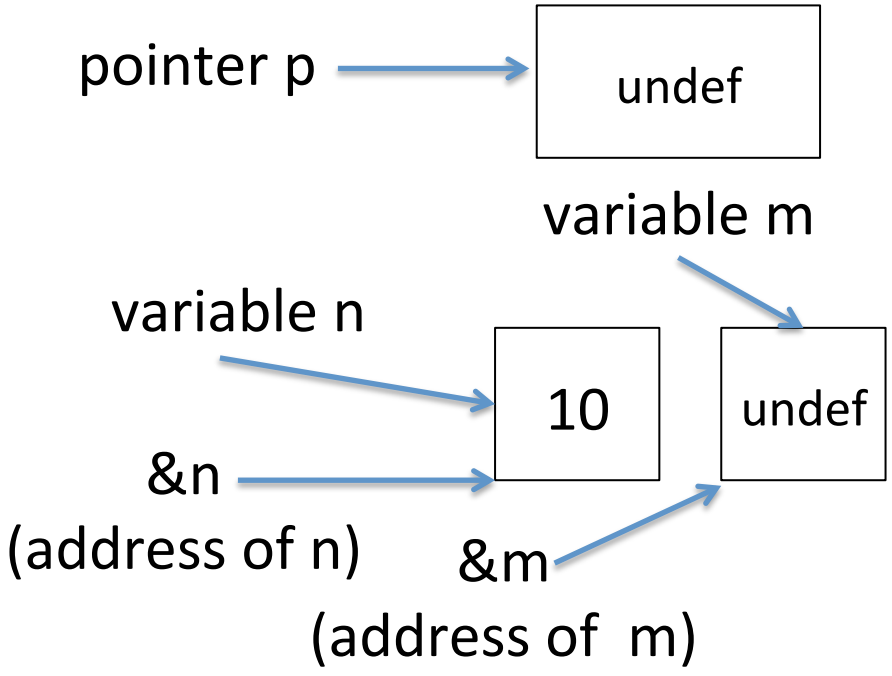
&: address of a pointer

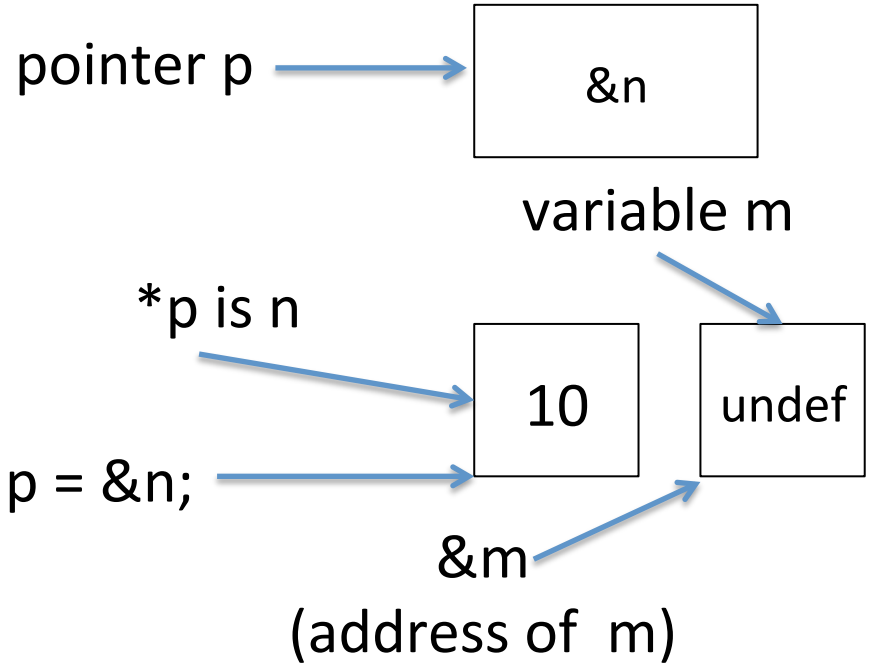
```
int n = 10;
int a[3] = {3,5,7};
```



Pointer

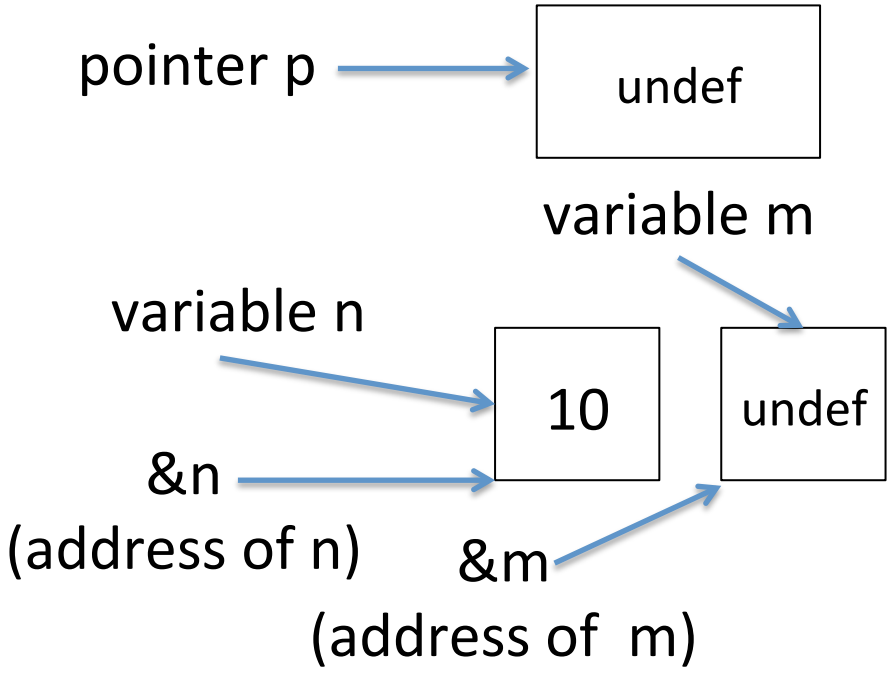
```
short int *p; // declares a pointer of type short int
short int m, n = 10; //variables of type short int

p = &n // the memory address of n is stored in p
m = *p // the contents of variable m is the value of
       // of the memory cell referenced by p
```

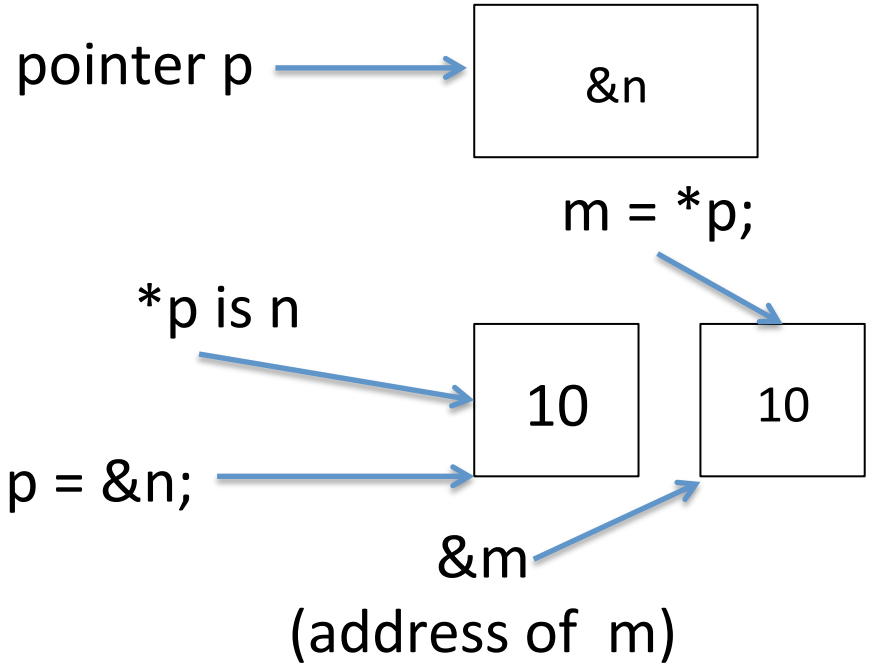




Important remark:

- ✧ **for a pointer one does not allocate memory space to store a value!!!**
- ✧ **you only allocate space for the pointer**





Pointer

```
int *p;
 *p = 17;
```

// This is commonly a mistake,
// rather than wanted!

- The above statement does not produce an error, but it is incorrect: we did not allocate space for storing an integer value
- p is undefined, i.e., it has an arbitrary value and therefore points to an arbitrary memory location
- the 17 will therefore be written to that (arbitrarily picked) location

```
int *p, n = 10;
p = &n;
*p = 17;
```

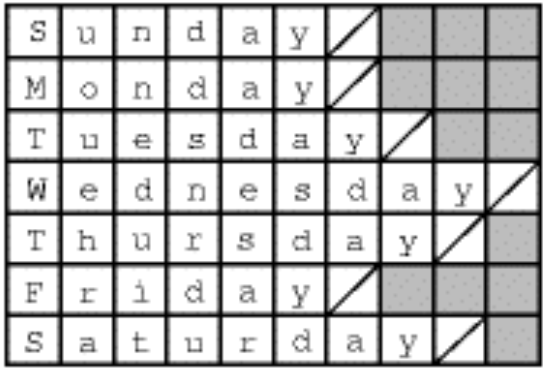
// What do we see here?

Multi-dimenisonal Arrays

Arrays can have multiple dimensions (n x m)

```
#include<stdio.h>

int main(){
   int a[4][4], i , j;

   for (i = 0; i < 4; i++){
      for ( j = 0; j < 4; j++){
         a[i][j] = 0;
         printf("a[%d][%d] = %d \n", i, j, a[i][j]);
      }
   }
   return 0;
}
```

```
char days[][10] = {
    "Sunday", "Monday", "Tuesday",
    "Wednesday", "Thursday",
    "Friday", "Saturday"
};
...
days[2][3] == 's';  /* in Tuesday */
```




```
char *days[] = {
    "Sunday", "Monday", "Tuesday",
    "Wednesday", "Thursday",
    "Friday", "Saturday"
};
...
days[2][3] == 's';  /* in Tuesday */
```