Arrays, Pointers and Memory Management

EECS 2031

Summer 2014

Przemyslaw Pawluk

May 20, 2014

Answer to the question from last week

`strct->field`

Returns the value of `field` in the structure pointed to by `strct` .

```
struct person {
                    int age, salary;
                    DEPT department;
                    char name[12];
                    char address[6][20];
                };
typedef struct person EMPLOYEE;

EMPLOYEE *Emp;
...
printf("Age: %d\n",Emp->age);
```

What we will discuss today

- Arrays
- Pointers
  - Pointers
  - Pointers and Arrays
- Memory management
- Homework

Table of Contents

Arrays

Pointers
- Pointers
- Pointers and Arrays

Memory management

Homework

Array

What is an Array?

- is grouping of data of the same type
- is continuous block of memory
- programmers set array sizes explicitly
- can be multidimensional (defined as array of arrays)
- loops are commonly used to manipulate data

Example

Syntax

```
type name[size];
```

Declaration examples

```
int bigArray[10]; //array of ints of size 10
double a[3]; //array of doubles of size 3
char grade[10], oneGrade; //chars of size 10
char *strings[]; //array of strings
int multiArray[][]; //two dimensional
```

Example

Definition of the array allocates the memory

`type arrName[size]`

Individual elements can be extracted/called

Using and *index* called also a *subscript* you can access array elements

Example

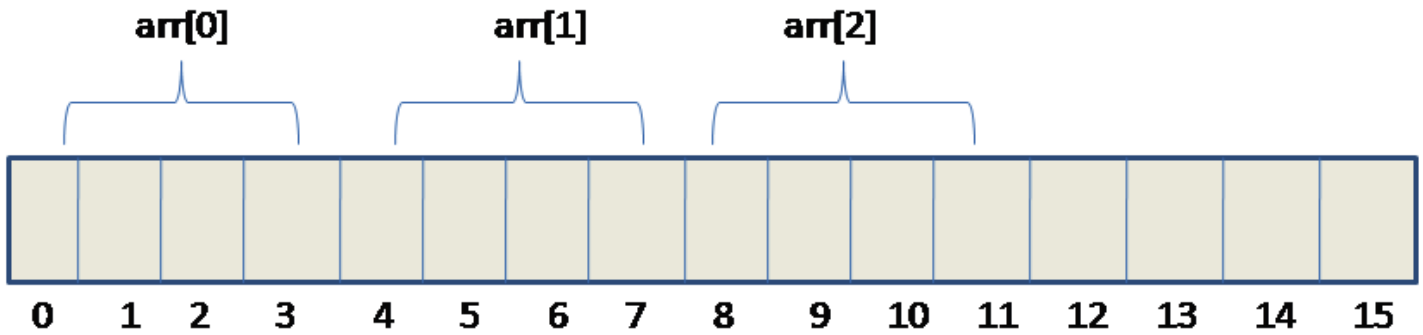
```
int score[5]; /*Allocates five (5) adjacent
                blocks of memory for int */

score[0] = 100; /*sets a value for first
                  element in the array*/
```

Array in the memory

Example

```
int arr[3] = {1,2,3};
```



- single block is a byte
- each integer occupies 4 bytes in the memory
- first element (index 0) stretches from byte 0 to byte 3
- Note: It is an example, in the memory addresses will not start from 0!

Initialization

Static variables

- `int a[5] = {22,51}` – declares array and init first two elements, remaining elements are 0
- `int b[] = {1, 3, 22, 51, 4}` – b has 5 elements listed in curly brackets

If the initialization is empty, elements are initialized with value 0 (for a specific type)

Examples

- `int c[10];` – array consisting 10 values 0
- `float d[10];` – array consisting 10 values 0.0f
- `char *str[5];` – array of 5 `NULL`

Arrays and indexes

Syntax

```
arrayName[index]
```

Examples

```
x = ar[2];
ar[3] = 2.7;
```

Question

What is the difference between the following expressions?

`ar[i]++` – increment value of i^{th} element

`ar[i++]` – access i^{th} value and move index

`ar[++i]` – move index and access i^{th} value

Arrays and indexes

Syntax

```
arrayName[index]
```

Examples

```
x = ar[2];
ar[3] = 2.7;
```

Question

What is the difference between the following expressions?

`ar[i]++` – increment value of i^{th} element

`ar[i++]` – access i^{th} value and move index

`ar[++i]` – move index and access i^{th} value

Arrays and indexes

Syntax

```
arrayName[index]
```

Examples

```
x = ar[2];
ar[3] = 2.7;
```

Question

What is the difference between the following expressions?

`ar[i]++` – increment value of i^{th} element

`ar[i++]` – access i^{th} value and move index

`ar[++i]` – move index and access i^{th} value

Arrays and indexes

Syntax

```
arrayName[index]
```

Examples

```
x = ar[2];
ar[3] = 2.7;
```

Question

What is the difference between the following expressions?

`ar[i]++` – increment value of i^{th} element

`ar[i++]` – access i^{th} value and move index

`ar[++i]` – move index and access i^{th} value

Arrays and indexes

Syntax

```
arrayName[index]
```

Examples

```
x = ar[2];
ar[3] = 2.7;
```

Question

What is the difference between the following expressions?

`ar[i]++` – increment value of i^{th} element

`ar[i++]` – access i^{th} value and move index

`ar[++i]` – move index and access i^{th} value

Table of Contents

Arrays

Pointers
- Pointers
- Pointers and Arrays

Memory management

Homework

Variables, Pointer and Addresses

Variable

Variable is a named block of memory of a specific type e.g. `int x;` or `float y;` or `char* str;`

Pointer

Pointer is a type of variable which value is an address of a variable in the memory,

Syntax: `type * variableName;`

Example: `char* str;` or `int* iptr;` or `float* fptr;`

Address

Address is a number representing an index of the first memory block of the variable (where does it start in the memory)

Use **address of** operator to get the address of the variable

`pointer_variable = &ordinary_variable`

Variables, Pointer and Addresses

Variable

Variable is a named block of memory of a specific type e.g. `int x;` or `float y;` or `char* str;`

Pointer

Pointer is a type of variable which value is an address of a variable in the memory,

Syntax: `type * variableName;`

Example: `char* str;` or `int* iptr;` or `float* fptr;`

Address

Address is a number representing an index of the first memory block of the variable (where does it start in the memory)

Use **address of** operator to get the address of the variable

`pointer_variable = &ordinary_variable`

Variables, Pointer and Addresses

Variable

Variable is a named block of memory of a specific type e.g. `int x;` or `float y;` or `char* str;`

Pointer

Pointer is a type of variable which value is an address of a variable in the memory,
Syntax: `type * variableName;`
Example: `char* str;` or `int* iptr;` or `float* fptr;`

Address

Address is a number representing an index of the first memory block of the variable (where does it start in the memory)
Use **address of** operator to get the address of the variable
`pointer_variable = &ordinary_variable`

Table of Contents

Arrays

Pointers

- Pointers
- Pointers and Arrays

Memory management

Homework

Pointers and Operators

- & operator – address of variable
- * operator – value pointed by the pointer

Example:

```
int *p1, v1;
v1 = 0;
p1 = &v1;
*p1 = 42;
printf( %d\n ,v1);
printf( %d\n,*p1);
```

Result:

```
42
42
```

Note

Assigning a value for a pointer variable manually is a bad idea! e.g.

```
int *p1;
p1=2341
```

Pointers and Function Arguments

C passes arguments to functions by values

Question: What is the difference between these two functions?

Function 1

```
void swap(int x, int y)
{
    int temp;
    temp = x;
    x = y;
    y = temp;
}
```

Function 2

```
void swap(int *px, int *py)
{
 int temp;
 temp = *px;
 *px = *py;
 *py = temp;
}
```

Table of Contents

Arrays

Pointers

- Pointers
- Pointers and Arrays

Memory management

Homework

Pointers and Arrays

Identifier of an array is equivalent to the address of its first element.

```
int numbers[20];
int *p;

p = numbers; // Valid
numbers = p; // Invalid
```

Arrays and Pointer Arithmetics

Array can be iterated using a pointer and * operator

Example

```
int a[] = {1,2,3,4,5,6,7,8,9,10};

int *pa;
int x, y, z;

pa = &a[0]; //a=pa is illegal;
            //a++ is illegal
            //but pa++ is OK
x = *pa; //x==1


y = *(pa + 1); //y==2
z = *(pa + 2); //z==3
```

Pointer Arithmetics cont.

Given pointers `p` and `q` of the same type and integer `n`, the following pointer operations are legal:

- `p + n`, `p  n`
- `n` is scaled according to the size of the objects `p` points to. If `p` points to an integer of 4 bytes, `p + n` advances by `4*n` bytes.
- `q  p`, `q  p + 10`, `q  p + n` (assuming $q > p$)
- But `p + q` is illegal!
- `q = p; p = q + 100;`
  - If `p` and `q` point to different types, must cast first. Otherwise, the assignment is illegal!
- `if ( p == q )`, `if ( p = q + n )` !
- `p = NULL;`
- `if ( p == NULL )`, same as if (!p)

Pointer Arithmetics – Summary

Legal:

- assignment of pointers of the same type
- adding or subtracting a pointer and an integer
- subtracting or comparing two pointers to members of the same array
- assigning or comparing to zero (`NULL`)

Illegal:

- add two pointers
- multiply or divide or shift or mask pointer variables
- add float or double to pointers
- assign a pointer of one type to a pointer of another type (except for `void *`) without a cast

Pointer Arithmetics – Summary

Legal:

- assignment of pointers of the same type
- adding or subtracting a pointer and an integer
- subtracting or comparing two pointers to members of the same array
- assigning or comparing to zero (`NULL`)

Illegal:

- add two pointers
- multiply or divide or shift or mask pointer variables
- add float or double to pointers
- assign a pointer of one type to a pointer of another type (except for `void *`) without a cast

Sub-arrays

Pointer can be used to access a sub-array and pass it to a function

Examples

```
my_func( int ar[ ] ) {...}
```
or
```
my_func( int *ar ) {...}
```

```
my_func(&a[5]);
```
or
```
my_func(a + 5);
```

Character Pointers

- A string constant (`"hello world"`) is an array of characters.
- The array is terminated with the null character `'\0'` so that programs can find the end.

Example

```
char *pmessage;
pmessage = "now is the time";
```

Assigns to `pmessage` a pointer to the first character in the array. This is not a string copy; only pointers are involved.

What is the difference?

```
char amessage[] = "now is the time";
char *pmessage = "now is the time";
```

- amessage will always refer to the same storage.
- pmessage may later be modified to point elsewhere.

Note

C does not provide any operators for processing an entire string of characters as a unit.

What is the difference?

```
char amessage[] = "now is the time";
char *pmessage = "now is the time";
```

- amessage will always refer to the same storage.
- pmessage may later be modified to point elsewhere.

Note

C does not provide any operators for processing an entire string of characters as a unit.

What is the difference?

```
char amessage[] = "now is the time";
char *pmessage = "now is the time";
```

- amessage will always refer to the same storage.
- pmessage may later be modified to point elsewhere.

Note

C does not provide any operators for processing an entire string of characters as a unit.

Table of Contents

Arrays

Pointers
- Pointers
- Pointers and Arrays

Memory management

Homework

Problem – following is not allowed in C

```
int x = 10;
int my_array[ x ];
```

How to allocate memory during run time?

Use functions to allocate (reserve) memory on heap,
e.g. `malloc()`, `calloc()`, `realloc()`

Problem – following is not allowed in C

```
int x = 10;
int my_array[ x ];
```

How to allocate memory during run time?

Use functions to allocate (reserve) memory on heap,
e.g. `malloc()`, `calloc()`, `realloc()`

malloc()

- Defined in `stdlib.h`
- Signature – `void *malloc( int n );`
- Allocates memory at run time.
- Returns a pointer (to a void, `void*`) to at least n bytes available.
- Returns null if the memory was not allocated.
- The allocated memory is not initialized.

calloc()

- Defined in `stdlib.h`
- Signature – `void *calloc( int n, int s );`
- Allocates an array of `n` elements where each element has size `s`
- `calloc()` initializes the allocated memory all to 0.

realloc()

- Defined in `stdlib.h`
- Signature – `void *realloc( void *ptr, int n );`
- Resizes a previously allocated block of memory.
- `ptr` must have been returned from a previous `calloc`, `malloc`, or `realloc`
- The new array may be moved if it cannot be extended in its current location.

free()

- Defined in `stdlib.h`
- Signature – `void free( void *ptr );`
- Releases the memory we previously allocated.
- ptr must have been returned from a previous `calloc`, `malloc`, or `realloc`.

Note:

C does not do automatic "garbage collection"

Example

```
#include<stdio.h>
#include<stdlib.h>
int main() {
  int *a, i, n, sum = 0;
  printf( "Input an array size: " );
  scanf( "%d", &n );
  a = malloc ( n * sizeof(int) );
  /* a = calloc( n, sizeof(int) ); */
  for( i=0; i<n; i++ )
      scanf( "%d", &a[i] );
  for( i=0; i<n; i++ )
     sum += a[i];
  free(a);
  printf("Number of elements = %d\n", n);
  printf("Sum is %d\n", sum);
  }
```

Table of Contents

Arrays

Pointers
- Pointers
- Pointers and Arrays

Memory management

Homework

Homework

Create a simple C program that:

- Reads a file called `input.txt` that contains a single integer number. Let's call it `N`
- Allocate an array of `char` of size `N`
- Reads `N-1` characters from input
- Saves them at the end into the file `output.txt`
- Frees the memory allocated for the array