

# C programming basics

Lecture 7

# Schedule

- Pointers
- Structures

Any questions ?

Homework solving

Questions ?

# Memories

- In a program there are 2 main types of memories:
  - Stack
  - Heap

# Stack

- Very fast memory that is used to initialize and use short-life variables

# Heap

- Slower memory used for large and long-life variables



# Variable addresses

- Each variable has its own place in the physical memory. It has its own cell(cells).
- This cell is initialized(allocated) with the variable and is used by this variable, until you stop using the variable (deallocate)

# Variable addresses

- How to get the address of a variable

```
int myVar = 8;  
printf("%x\n", &myVar);
```

# Variable addresses

- Using the ‘&’ character of a variable, we can read the physical address of a variable.

Questions ?

# Pointers

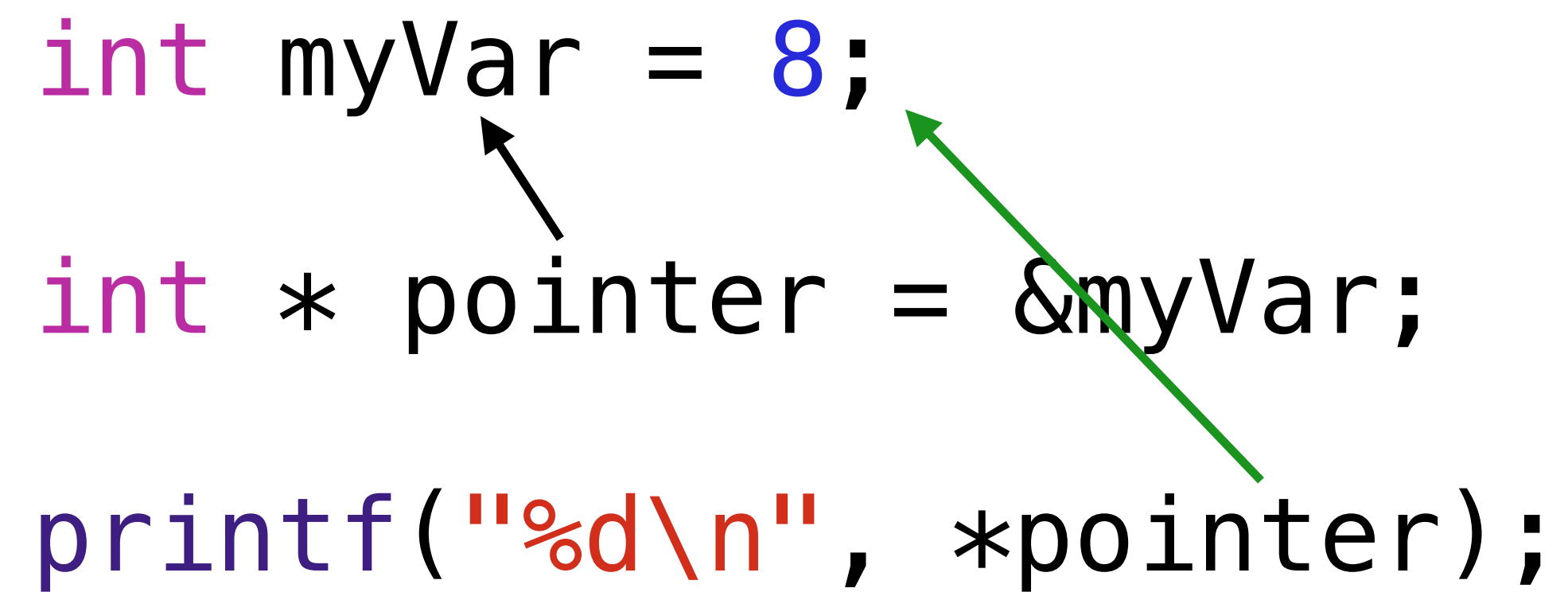
- So, is it possible to have an “alias” for a variable, and if we change the value of this alias to change the value of the variable ?
- How to do that ?

# Pointers

- In programming there is a specific type of variable, called pointer. The pointer is “pointing” to a cell in the memory. If we change the pointer we can set it to point to other cell, if we change the value of the pointer, we can change the value of the cell.

# Pointers

```
int myVar = 8;  
int * pointer = &myVar;  
printf("%d\n", *pointer);
```



The diagram illustrates the relationship between the code and memory. A black arrow points from the `&` operator in `&myVar` to the `myVar` variable in the first line. A green arrow points from the `*` operator in `*pointer` to the `myVar` variable in the first line, indicating that the pointer variable `pointer` holds the address of `myVar`, and `*pointer` dereferences it to access the value stored in `myVar`.

# Pointers

- And there comes the question, how using the pointer, we can change the value which it points to ?



# Pointers

```
int * pointer = &myVar;  
*pointer = 9;  
printf("%d\n", myVar);
```

# Pointers

- Change the variable the pointer points

```
int myVar = 8;  
int nextVar = 18;  
int * pointer = &myVar;  
pointer = &nextVar;  
printf("%d\n", *pointer);
```

Questions ?

# Memory allocation

# Memory allocation

- When we must initialize arrays, string or any large variables, it is good to have them initialized in the heap memory and have a pointer which is pointing there.

# Memory allocation

- The easiest way to allocate a memory is using the malloc (memory allocate) function. It needs as argument the number of **bytes** that you need to allocate.

# Memory allocation

```
char *string = malloc(20); // we initialize 20 bytes
in heap
for(int i = 0; i < 20; i++){
    string[i] = rand()%26 + 'a';
}
string[20] = 0;
printf("%s\n", string);
```

# Caution!

- After you allocate and use memory in your app, you must free this memory, when you are done using it. For the stack variables this happens automatically, but for the heap variables, you must do it by yourself !
- This is done by calling `free(pointer)` function.



# Freeing memory

```
char *string = malloc(20); // we initialize 20 bytes  
in heap  
free(string); // we free this memory
```

# Freeing memory

- When you forget to free this memory, it is called memory leak, and may result in memory overflow at some point of using the app.

Questions ?

Some useful functions

# memset(pointer, value, size)

- Memset is used to set many blocks of memory with one value. It is good for strings when you want to clear all characters in string.

```
char *string = malloc(20); // we initialize 20 bytes  
in heap  
memset(string, 0, 20);
```

# strcpy(dest, source)

- String copy is used to copy characters from one string to another  
    char \*string = malloc(20); // we initialize 20 bytes  
in heap  
    memset(string, 0, 20);  
    strcpy(string, "Hello world");

# strlen(pointer)

- strlen is used to check the number of length of a string

```
strcpy(string, "Hello world");
unsigned int length = strlen(string);
printf("%u", length);
```

# strcmp(str1, str2)

- strcmp is used for string compare. If it return 0 - the strings are equal

```
strcpy(string, "Hello world");  
int notEqual = strcmp(string, "Hello World");  
if (notEqual == 1) printf("Not equal");  
if (notEqual == 0) printf("Equal");
```



# strcat(str1, str2)

- Strcat is used to append string1 with the characters of string2

```
char *string = malloc(20);  
memset(string, 0, 20);  
strcat(string, "Hello ");  
strcat(string, "my world");
```

# sizeof(type)

- sizeof(type) is used to check the size of bytes that a certain variable type needs to be initialized in the memory.

```
printf("%d", sizeof(char));
```

Questions ?

# Structures

# Structures

- Structures are used to encapsulate many properties that are common for one object. For example if you have a car, and this car have properties - make, model, horse power, you can make the following structure:

# Structures

```
typedef struct{  
    char * make;  
    char * model;  
    unsigned int horsepower;  
} Car;
```

```
Car myCar;
```

# Structures

```
int main(int argc, const char * argv[]) {  
  
    myCar.make = "Honda";  
    myCar.model = "Civic Type R";  
    myCar.horsePower = 198;  
    printf("%s", myCar.make);  
  
    return 0;  
}
```

# Structures

- The structs are ultimately powerful, because you can set them as argument in functions and do required actions. You can make array of structures and keep a local copy of a “data base” and use changes as they are needed.



# Structures

```
void printFormattedCarInfo(Car car){  
    printf("Make: %s\n", car.make);  
    printf("Model: %s\n", car.model);  
    printf("HP: %u\n", car.horsePower);  
}
```

# Structures

```
int main(int argc, const char * argv[]) {  
    Car myCar;  
    myCar.make = "Honda";  
    myCar.model = "Civic Type R";  
    myCar.horsePower = 198;  
    printFormattedCarInfo(myCar);  
    return 0;  
}
```

# Daily task

- A professional car garage needs a system that is used to present data about the horse power and torque of cars in the cases when the cars are with original ECU settings and when they are tunes.
- The system must be able to search between all the cars based on the car model and present all data.
- The system must be able to present the car with most HP, the car with most Torque, when it is tuned.
- The system works with 10 cars (for now). It must be able to work with more than 100. It is good to be able to work with unlimited amount of cars.