Metropolia University of Applied Sciences

Programming
TI00AA43-3003
Lecture 3

Sami Sainio
sami.sainio@metropolia.fi

Metropolia

Plan for going forward

- Variables and printing to screen
- If, else, while and for loops
- **I/O**
- Functions and tables
- File handling
- Pointers and arrays
- Simple structures
- Program structure and design

I/O

I/O (input/output, reading/printing)

- Stdio models character and text input/output (I/O)
  - I/O is a stream of characters that are on one or multiple lines
- Reading / writing characters one character at a time
  - cha = getchar(); // int cha;
  - putchar(cha); // int cha;
- getchar() return either ASCII code from the character read or constant EOF (defined in stdio.h)
- EOF marks the end of the stream

Difference between getchar() and scanf()

- What are the major differences between getchar() and scanf()?
- By using goodle, you have 15 minutes to find out how they work!

Getchar()

- By calling the funtion getchar() single character from the stdin buffer will be read
- cha = getchar()
- The variable that was in the stdin buffer will now reside in cha variable
- Main point is to realize it is possible to read one character at a time (per call of getchar())

Scanf()

- Scanf() will read a string and can place the input to a single variable or to an array
- Scanf() function is defined so that by default it will terminate to a whitespace scaracter, newline character and so on
- What does this mean?

Scanf() continued

- If it is necessary to read ”hello world” from the terminal it is necessary to use either two variables, or tell scanf() funtion not to terminate on the condition whitespace:
  - char a[10];
- scanf("%10[^\n]s", a);
  - 10 = stop at 10 characters
  - [^\n] = stop at newline character
  - s = string

Input of unknown size (files especially)

- I recommend using getchar()
- Why?
- More convinient on the actual reading, it doesn’t matter what we are reading, we will terminate on given condition, not already defined condition! (scanf() had whitespace as default)

Functions and tables

Functions

- So far we've used plenty of functions such as
  - printf(), scanf(), getchar(), putchar()

- Functions are very helpful when programming, function that is done and implemented well once is reusable in the future!

```
#include <stdio.h>

int squared(int target);

int main(void) {
        int i;
        for(i=0;i<10;i++) {
                printf(“%d\t squared is %d\n”, i, squared(i));
        }
return 0;

}

int squared(int m) {
        int p;
        p=m*m;
        return p;
}
```

Prototype of the function, parameter: int type variable

Function call

Actions taken in the function

Function: observations 1/2

- Prototype of the function must appear before the function is called for the 1st time
- Function can be defined before or after the function is called
- Arguments for the function are passed as values!

Function: observations 2/2

- Changing the parameters of the function inside the function will not change them outside the function!
- Parameters for the function are *private*
- Difference: arrays. Their arguments are pointers to the first cell location of the array (memory location). More about this when we discuss pointers.

```
#include <stdio.h>

void printing(int f_age, int f_height);

int main(void)
{
    int age, height;
    printf(”Age and height, separated by space:");
    scanf("%d %d", &age, &height);
    printing(age, height);
    return 0;
}

void printing(int fin_age, int fin_height)
{
        printf(”Printing from function!\n");
        printf(”Age %d, height %d\n", fin_age, fin_height);
}
```

Prototype of the function, two parameters that are int type

Function call

Actions taken inside the function

Global variables

- When using functions (and why not in other cases as well) it might be necessary to use global variables
- Global variable exists in main as well as in functions. If it is modified anywhere its value will remain modified even when returning from function for example

```
#include <stdio.h>
int g_age;

void printing(int f_height);

int main(void)
{
    int age, height;
    printf(”Age and height, separated by space:");
    scanf("%d %d", &g_age, &height);
    printing(height);
    printf(“Reprint global variable: %d\n”, g_age);
    return 0;
}

void printing(int fin_height)
{
        printf(”Printing from function!\n");
        printf(”Age %d, height %d\n", g_age, fin_height);
        printf(“Reprint global varible %d\n”, g_age);
        g_age++;
}
```

Declare global variable
int g_age

Read user input

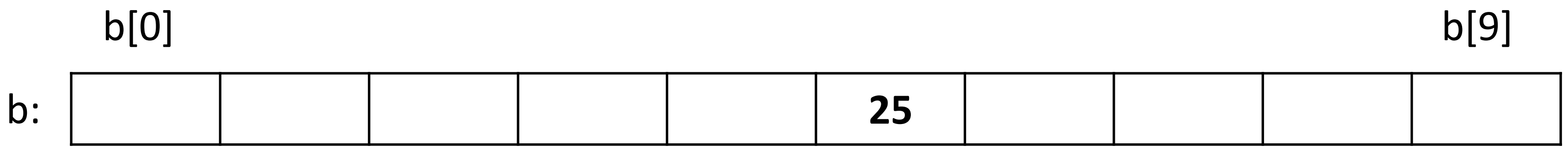
Print and incrase global variable

Arrays

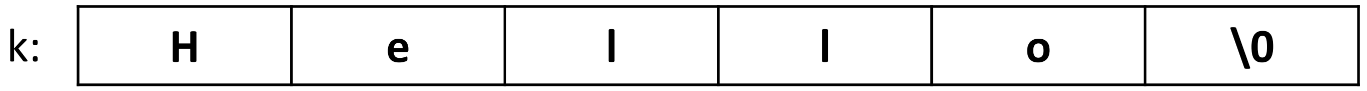
- Array is declared as follows:
- int b[10]; //declares array that has 10 int type cells
- Indexing of the arrays starts at 0 and ends at length-1 (in this case 0-9 are array indexes)

Arrays

```
int b[10];
b[5] = 25;
```



```
char k[] = ”Hello”;
```



Arrays

- Character array is terminated with \0 by compiler in C

- This is a feature that you should not fight against

What does the following code do?

```
#include <stdio.h>
int main(void)
{
		char ch_array[] = "Hello, world";
		int i;
		for(i = 0; ch_array[ i ] != '\0'; i++)
		      ;
		printf(“String %s lenght is %d\n", ch_array, i);
		return 0;
}
```

Reading input to an array

- Reading int type values to array is easily done by increasing the index by one as new values arrive
- Programmer must take care that user is not able to give too many values!

```
#include <stdio.h>
#define MAX 10
int main(void) {
        int num, i;
        int cnt = 0;
        int sum = 0;
        int allNum[ MAX ];
        do {
                printf(”Give integer (negative ends): ");
                scanf("%d", &num);
                if(num >= 0) {
                        allNum[ cnt++ ] = num;
                }
        } while ((num>= 0) && (cnt < MAX));

        for(i = 0; i < cnt; i++) {
                sum = sum + allNum[ i ];
        }
        printf(”Thanks. Sum is %d\n", sum);
        return 0;
}
```

Declare global constant

Read user input

Count sum from user input values