Programming in C

Week 2

9.9.2015

Tiina Niklander
Meeting structure

- First week
  - Some notes

- Second week
  - Focus on pointers

- Slides related to first week topics – covered if time allows or some questions arise
First week tasks

- TMC problems
  - Some tests did not accept correct answers on the server
  - Difficulties configuring NetBeans properly

- Tasks
  - Uninitialized values: test failure information not useful
  - Printf: formatting problems, especially \n
Briefly about pointers

char *p;  /* pointer to a character or string */
int *q;  /* pointer to one integer (or array) */
/*Memory allocated only for the pointer! */

char *p = "This string is allocated";
int numbers[] = {1, 2, 3, 4, 5};
double table[100];
Allocate memory for the array and set the pointer to the array.
(No memory allocated for array name "constant pointers", only allocates the memory block containing the values!)
Pointers (and arrays)

- Array is just a sequence of values with a joint name.  
  int a[15] is sequence of 15 integers.
- Array name is treated as a pointer, whose value is the address of the first element in the sequence.  
  pa = &a[0]  
  pa = a
- Pointer arithmetic allows operations on array elements  
  *(pa +3) is the same as  a[3]  
  pa+3  is the same as  &a[3]
**Pointer arithmetics and operations**

\[ p = \&c \] address of \( c \)

\[ c = \ast p \] value of the address pointed by \( p \)

\[ c = \ast \ast r \] \( -"- \) (two 'jumps')

\[ p = q \] allowed when \( p \) and \( q \) of same type

\[ p+i, \ p-i \] \( p \) is array, \( i \) has to be integer with suitable value

\[ p-q, \ p < q, \ p == q \] \( p \) and \( q \) pointers of the same array and \( q<p \)

\[ \ast ip++ \] increments the address by 'one'

\[ (\ast ip)++ \] increments the value in the address by one

**Remember:**

NULL
int **p;
int *q, r;
int i;
i = **p
q = &i; /* q's new value is i's address
i = *q + 1;   /* i = 6 */
i = *q ++; /* ???? */
r = q; *r = 3; /* i = 3 */

void *p; i = *(int*) p;

const int *p;
int const *p;
const int const *p;

char msg[] = "It is time"; msg: It is time\0
char *pv = "It is time"; pv: It is time\0
Example code

```c
int main(int argc, char** argv)
{
    int x=1, y=2, z[10];
    int *ip;
    int *p, q;
    int *r, *s;

    ip = &x;
    y = *ip; /* y = x = 1 */
    *ip = 0; /* x = 0 */
    ip = &z[0];
}

double atof(char * string);
```

Pointers as arguments for functions are very common. (Always used with arrays and needed for call by reference.)

p is a pointer variable and q is integer variable
Memory allocation

- Explicit memory allocations!
- malloc – static data structures
- calloc – dynamic array
- realloc – change the size of already allocated object
- free – deallocate the memory

/* ALWAYS CHECK THE RETURN VALUE!!! */
if (k=malloc(sizeof(double)))
    error; /* allocation failed, do something else or terminate program */

/* memory allocation succeeded and k is the pointer to the new structure */
Functions: Call by value, call by reference

C uses always call by value => function cannot change the value it receives as argument.

Call by reference done with pointers!!!

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

```c
void swap(int *x, int *y) {
    int apu;
    apu=*x;
    *x=*y;
    *y= apu;
}
```

void swap(int *x, int *y) {
    int apu;
    apu=*x;
    *x=*y;
    *y= apu;
}

Call: swap (&x, &y);

x | y
---|---
3  | 4

copies

double product (const double block, int size);

Make sure that function does not change the variable (ANSI standard!)
Example code: copy a string - Passing array to a function

```c
#include <stdio.h>

void copy_string( char *s, char *t)
{
    int i =0;
    while ( (s[i] = t[i]) != '\0' )
        i++;
}

int main (void)
{
    char here [] ="This string is copied.",
            there[50];
    copy_string ( here, there);
    printf("%s\n", there);
    copy_string ( there, here);
    printf("%s\n", there);
    return 0;
}
```

Strings (character arrays) as arguments. C is always passing only the address of the first element of any array.

Processing one character of each array
Example code: copy a string – Now with pointers

Version 1:

```c
void copy_string( char *s, char *t)
{
    while ( (*s = *t) != '\0' )
        s++; t++;
}
```

Version 2:

```c
void copy_string( char *s, char *t)
{
    while ( (*s++ = *t++) != '\0' )
        ;
}
```

Version 3:

```c
void copy_string( char *s, char *t)
{
    while ( *s++ = *t++ )
        ;
}
```

NOTE: The function prototype is identical with the previous slide

Minimalistic!
More about pointers and some good practices

- **Generic pointer** (void *p) can be used with type cast to handle a variable of that type.
  *(double *)p

- **Memory allocation** for n integers
  ```c
  int *p;
  if ((p=malloc(n*sizeof(int))) == NULL)
    error;
  ```

- **Memory deallocation**: remember to free(p); p=NULL;

- i’th element of array
  ```c
  p[i]  (preferred over *(p+i) )
  ```

- Handling an array p
  ```c
  for (pi = p;  pi < p+SIZE; pi++)
    remember to use pointer pi in the loop
  ```
Still more

- Call by reference
  1. Prototype’s argument – a pointer
     ```c
     void func(int *pp)
     ```
  2. In the function use the pointed value.
     ```c
     *pp
     ```
  3. In the function call: address of the variable
     ```c
     func(&variable);
     ```
  4. In the function call: pointer
     ```c
     func(pointer_variable);
     ```
- Array of struct
  ```c
  for (p = block; p < block + n*elSize; p+= elSize)
  ```
  i. element of struct array
  ```c
  p = block + i*elSize
  ```
Evaluation order
Precedence

Arithmetical ops
Bitwise moves
Value comparations

Bitwise comparations
and
or
Conditional op

Same line - same priority

( ) [ ] . - >
! ~ - ++ -- & * (tyyppi) sizeof
* / %
+ -
<< >>
< <= > >=
== !=
& ^
| ||
?:
= *= /= %= += -= <<= >>= &= != ^= ,
Errors
Assosiativity

Expression

\[ a < b < c \]

is interpreted as

\[ (a < b) < c \]

And the meaning is different than expression

\[ a < b \&\& b < c \]
Programming Guidelines

Style: using space

Do not use space with the following:
- -> . [] ! ~ ++ -- -(sign)
- *(pointer) &

Usually have space around these:
- = += ?: + < &&
- + (addition) and others

```
a -> b      a[i]    *c
a = a + 2;
    a = b + 1;
    a = a + b * 2;
```
Constants

Defined as variables, but with addition const

Usually constant names in capital letters

const float PI = 3.1412;
const int BIG_NUMBER = 0xFF7D;
const int TRUE = 1;
const int FALSE = 0;
const char LETTER_A = 'a';
const char [] MJONO = ”String has parenthesis around it”;
Macros

Preprocessor control – textual replacement!
Macro is a text that is replaced with other text before the actual compilation
NOTE: Whole end of the line is the replacement string as it is!!

Can be used to define ‘constants’ but is more powerful

```c
#define MAKSIMI 30
#define NAME “Tiina Niklander”
#define TRUE 1
#define FALSE 0
```
String vs character array

char letters[30];
char* char_pointer;

Array letters contains characters = character array

When the last character is \0 then considered as string
Errors
Avoid mistakes

- $i = 8$ different than $i == 8$

- Remember to set initial values to variables!
- Check the limits (avoid ‘off by one’)

- These are not logical operations!!
  
  ```
  e1 & e2
  e1 | e2
  if(x = 1) ...
  ```
NEVER test overflow with

\[ i + j > \text{INT\_MAX} \]

BUT do:

\[ i > \text{INT\_MAX} - j \]

Source: Müldner
Slides related to first week
## Simple types

<table>
<thead>
<tr>
<th>Data Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Int</td>
<td>28, 074, 0x2A</td>
</tr>
<tr>
<td>char</td>
<td>One character, actually a numerical value, do not assume anything</td>
</tr>
<tr>
<td></td>
<td>’a’ \065 \xA6</td>
</tr>
<tr>
<td>float, double</td>
<td></td>
</tr>
</tbody>
</table>

**NOTE:** no boolean
- Use integer values
- 0 - FALSE and all other values TRUE

Size of these not fixed between systems (see: sizeof or limits.h)

<table>
<thead>
<tr>
<th>Signed, Unsigned</th>
</tr>
</thead>
<tbody>
<tr>
<td>unsigned int</td>
</tr>
<tr>
<td>signed char</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Short, Long</th>
</tr>
</thead>
<tbody>
<tr>
<td>long char</td>
</tr>
<tr>
<td>short int</td>
</tr>
</tbody>
</table>

**Combined**
- signed short int
- unsigned long int
Header file: limits.h

#include <limits.h>

Limits.h contains the maximum and minimum values of different types in this environment.

At department the file is in /usr/include/

Always: INT_MAX >= 32767

Lots of values: eg. SHRT_MAX (signed short)
With ints you can define the type after value (U, L)

12U is unsigned int and 7L long int

sizeof(short) <= sizeof(int) <= sizeof(long)
Header file: float.h

#include <float.h>

Contains size and limit values for
- float
- double
- long double

sizeof(float) <= sizeof(double) <= sizeof(long double)
Type conversion

Implicit: operands with different types -> automatic type conversion for the arithmetic operation using the 'better quality' type:

- int ja char
- unsigned
- long
- unsigned long
- float
- double
- long double

Explicit:
(double)int_var;
(int) letter;
Statements

Conditional

Loops

Interrupting a loop

Break - continue from the statement AFTER the loop
Continue – continue with NEXT ROUND
Not named!!
Using break

```c
While (1) {
    printf("give two numbers a and b, a < b:"");
    if (scanf("%d%d", &a, &b) == 2)
        break;
    if (a < b)
        break;
    ...
}
/* break continues from here */
```

Several typical C features
- eternal loop while(1)
- error checks !!
- standard functions
Exiting from a deep loop structure

Exit over multiple loop levels must be done with goto (Avoid using for anything else!)

```
for(i = 0; i < length; i++)
  for(j = 0; j < length1; j++)
    if(f(i, j) == 0)
      goto done;

done:
```

Break would continue the outer loop!
/* Beginning of main and variable definitions */
Printf("Please give at most %d chars\n", LIMIT);
For (i = 1; i <= LIMIT; i++) {
    if ( (c=getchar()) == EOF)
        break;  /* end of file with CTRL-D */
    switch (c) {
    case ' ': space++;
        break;
    case '\t': tabul++;
        break;
    case '*': asterisk++;
        break;
    default : if (c>='a' && c<='z')
        lowercaseletters++;
        break;
    }
}
...  /* continues e.g. with printing */
/* Program that reads two integer values, and * outputs the maximum of these values. */

#include <stdio.h>

int main() {
    int i, j;

    printf("Enter two integers:");
    if(scanf("%d%d", &i, &j) != 2) {
        fprintf(stderr, "wrong input\n");
        return EXIT_FAILURE;
    }

    printf("Maximum of %d and %d is %d\n", i, j, i > j ? i : j);
    return EXIT_SUCCESS;
}
This loop

```plaintext
while(expr != 0)
    statement;
```

Is identical with this one

```plaintext
while(expr)
    statement;
```
while(1) {
    if((aux = getchar()) == EOF || aux == SENTINEL)
        break;
    ...
}

*or:*
while(1) {
    if((aux = getchar()) == EOF)
        break;
    if(aux == SENTINEL)
        break;
    if(aux == SENTINEL)
        break;
while(1) {
    if (scanf("%d", &i) != 1 ||
        i == SENTINEL)
    break;
    ...
    ...
Input and output briefly

Character at a time

```c
int getchar()
int putchar(int)
```

Formatted

```c
int scanf("format", &var)
int printf("format", exp)
```
/ File: ex1.c

* Program that reads a single character and
* outputs it, followed by end-of-line

#include <stdio.h>
#include <stdlib.h>

int main() {
    int c;  /* chars must be read as ints */
    if ((c = getchar()) == EOF)
        return EXIT_FAILURE;
    putchar(c);
    putchar('
');
    return EXIT_SUCCESS;
}
Printf & scanf: integer values

\texttt{d} \quad \text{signed decimal}
\texttt{ld} \quad \text{long decimal}
\texttt{u} \quad \text{unsigned decimal}
\texttt{o} \quad \text{octal}
\texttt{x, X} \quad \text{hexadecimal}

\texttt{printf("\%d\%o\%x", 17, 18, 19);}
Printf and scanf: real number, floating point numbers

default is 6 digits:

\( f \)  
\([-\] d\, \text{ddd. ddd} \\
\( e \)  
\([-\] d. \text{ddddde\{sign\}dd} \\
\( E \)  
\([-\] d. \text{ddddE\{sign\}dd} \\
\( g \)  
\( fe \) (\( f, e \) only if needed (e.g. sign \(<-4\))

\( G \)  
\( FE \)

\begin{verbatim}
printf("\%5.3f\n", 123.3456789);
printf("\%5.3e\n", 123.3456789);
123.346
1.233e+02
\end{verbatim}
Printf and scanf: chars and string

- c, one character
- s, string

```c
printf("%c", 'a');
printf("%d", 'a');
printf("This %s test", "is");
```
`scanf()` – return value

`scanf()` returns as its value the number of read items and `EOF`, if not item was read before the end-of-file occurred.

For example, `scanf("%d%d", &i, &j)` may return:

2  If both values were read correctly
1  If only i was read
0  If reading failed completely
`EOF` if file ended.