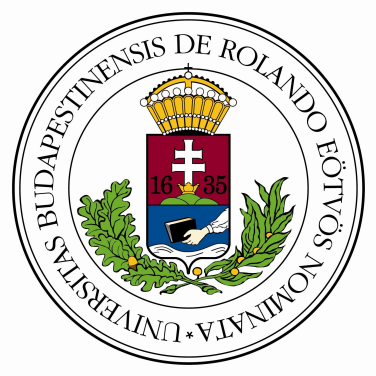
# Imperative programming

## 4th Lecture




**Kozsik Tamás**

ELTE Eötvös Loránd Tudományegyetem

# Outline

# Expressions - repetition

- Lexica: literal, identifier, operator, parentheses, comma
- Syntax: arity and fixity of operators
- Semantics: Process of Evaluation
  - Ordering
    - Precedence
    - Associativity
  - Side-effect
  - Greediness / Laziness
  - Operands / Evaluation Order of Actual Parameters
  - Sequence Point

```
i -- - -- i    /* undefined result, not determined outcome */
```

# Sequence Point (in C)

- End of an Expression
- End of evaluating the Actual Parameters of a Function
- End of evaluating the 1st operand of Lazy operators
- Comma operator

# Comma Operator (in C)

```
<expression> ::= ...
               | <expression> ,  <expression>
```

- Its result is the result of the rightmost expression
- Lowest precedence level

```
int i = 1, v;
v = (++i, i++);   /* not the same as: v = ++i, i++; */
```

# Comma: Operator or Separator (in C)

```c
int i = 1, v;
if( v = f(i,i), v > i )
    v = f(v,v), i += v;

for( i = f(v,v), v = f(i,i); i < v; ++i, --v ){
    printf("%d %d\n",i,v);
}
```

# Values

- Number
  - Integer (`144L`, `-23`, `0xFFFF`)
  - Floating-point (`123.4`, `314.1592E-2`)
  - Complex (`3.14j`)
- Logical value (`0` or `False`)
- Character (`'a'`, `'\n'`)
- Sequence
  - String
  - Value Sequence

# Characters

## C

A whole number.

- Character code on 1 byte, e.g. ASCII

```
char c = 'A';       /* ASCII: 65 */
```

Escape-sequences

- Special characters: `\n`, `\r`, `\f`, `\t`, `\v`, `\b`, `\a`, `\\`, `\'`, `\"`, `\?`
- Octal code: `\0` – `\377`
- Hexadecimal code, e.g. `\x41`

# Signed and Unsigned char (in c)

```
  signed char a = '\xFF';    /* a < 0          */
unsigned char b = '\xFF';    /* b > 0          */
         char c = '\xFF';    /* platform dependent */
```

# Wider Representation

```
wchar_t w = L'é';
```

- Implementation Dependent!
  - Windows: UTF-16
  - Unix: usually UTF-32
- from C99 „universal code'':, e.g. `\uC0A1` and `\U00ABCDEF`

# Text in C

- Not a `string`!
- Array of characters, terminated by `'\0'`
  - Indexed starting with 0

```
char word[] = "apple";
printf("%lu\n", sizeof(word));   /* 6 */

char a = word[0];
word[0] = 'A';

wchar_t wide[] = L"körte";
```

# Accented Characters in Text (in C)

- Platform dependent representation
- One character can be represented on multiple bytes
  - e.g. UTF-8

```
char word[] = "körte";
printf("%lu\n", sizeof(word));   /* 7 */
```

# Allocating an Array of Characters

```c
char w1[] = "alma";
char w2[8] = "alma";
printf("%lu %s\n", sizeof(w1), w1);    /* 5 alma */
printf("%lu %s\n", sizeof(w2), w2);    /* 8 alma */
```

# Danger: Allocating Too Small Array

```c
char w1[] = "lakoma";
char w2[4] = "alma";
printf("%lu %s\n", sizeof(w1), w1);    /* 7 lakoma */
printf("%lu %s\n", sizeof(w2), w2);    /* 4 almalakoma */
```

# Zero inside a Text

```c
char word[] = "lak\0ma";
printf("%lu %s\n", sizeof(word), word);    /* 7 lak */
printf("%c\n", word[4]);                   /* m */
```

# Manipulating Text

```c
#include <string.h>
#include <stdio.h>

int main()
{
    char word[100];
    strcpy(word,"alma");
    strcat(word,"lakoma");
    printf("%lu %s\n", sizeof(word), word);  /* 100 almalakoma */
    printf("%lu\n", strlen(word));           /* 10 */
    return 0;
}
```

# Outlook

```c
char w1[]  = "alma";   /* text is inside it */
char w2[6] = "alma";   /* text is inside it */
char * w3  = "alma";   /* points to text, shouldnt be modified */
printf("%lu %s\n", sizeof(w1), w1);   /* 5 alma */
printf("%lu %s\n", sizeof(w2), w2);   /* 6 alma */
printf("%lu %s\n", sizeof(w3), w3);   /* 8 alma */

w1[0] = 'A';
w2[0] = 'A';
w3[0] = 'A';      /* problematic - Segmentation Fault? */
```

# Type of Sequences

- C: array

# C array

```c
double point[3];      /* size should be constant */
point[0] = 3.14;      /* indexed from zero */
point[1] = 2.72;
point[2] = 1.0;
```

# Initialization of C Array

```c
double point[] = {3.14, 2.72, 1. + .1};
            /* elements should be constant if in global scope */


point[2] = 1.0;    /* can be modified */
```

# Processing

```
#define DIMENSION 3

double sum( double point[] ){
    double result = 0.0;
    int i;
    for( i=0; i<DIMENSION; ++i ){
        result += point[i];
    }
    return result;
}

int main(){
    double point[DIMENSION] = {3.14, 2.72, 1.0};
    printf("%f\n", sum(point));
    return 0;
}
```

# Generalization

```
double sum( double nums[], int length )
{
    double result = 0.0;
    int i;
    for( i=0; i<length; ++i ){
        result += nums[i];
    }
    return result;
}

int main()
{
    double point[] = {3.14, 2.72, 1.0};
    printf("%f\n", sum(point,3));
    return 0;
}
```

# Hazards

## Compilation Error

```
double point[DIMENSION] = {3.14, 2.72, 1.0, 2.0};
```

## Uninitialized Elements

```
double point[DIMENSION] = {3.14, 2.72};
```

## Over-indexing, illegal memory read

```
printf("%f\n", point[1024]);
```

## Over-indexing, illegal memory write (buffer overflow)

```
point[31024] = 1.0;   /*  Segmentation fault? */
```

# Texts as Arrays


```
char good[] = "good";
char  bad[] = {'b', 'a', 'd'};
char ugly[] = {'u','g','l','y','\0'};
printf("%s %s %s\n", good, bad, ugly);
```