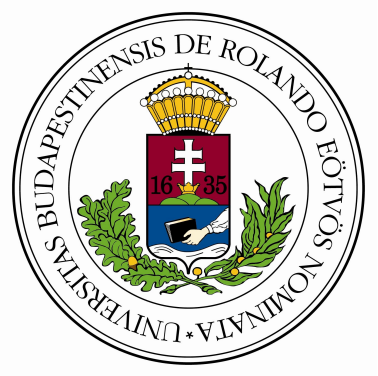
Imperative programming

10th Lecture



Kozsik Tamás

ELTE Eötvös Loránd Tudományegyetem

Outline

Function Declarations and Definitions in C

```c
int f( int n );
int g( int n ){ return n+1; }

int h();
int i(void);

int j(void){ return h(1); }

int h( int p, int q ){ return p+q; }

extern int k(int,int);

int printf( const char* format, ... );
```

Function Definitions in Python: Variable Parameter List

```
def sum( *args ):
    s = 0
    for n in args:
        s += n
    return s

sum()
sum(3)
sum(3,2)
sum(3,2,7,6,1,8)
```

Function Definitions in Python: Parameter Passing Denoted by Name

```
def copy( src, dst ):
    for item in src:
        dst += [item]

a = [1,2,3]
b = [4,5]
copy( dst=b, src=a )
```

Function Definitions in Python: Default Value of Parameter

```
def unwords( words, separator=' ' ):
    length = len(words)
    if length == 0:
        return ''
    else:
        result = words[0]
        for i in range(1,length):
            result += separator
            result += words[i]
    return result

unwords(["apple","under","the","tree"])
unwords(["apple","under","the","tree"], separator='\n')
unwords(["apple","under","the","tree"], '\n')
unwords(separator='\n', words=["apple","under","the","tree"])
```

Outline

Concept of Arrays

Object of the same type (similar sized) sequentially in memory.

- Any of them can be accessed efficiently.
- Fixed number of objects!

```
int vector[4];
int matrix[5][3];   /* 15 elements continuously */
```

Indexed from 0

- address of `vector[i]`: address of `vector + i * sizeof(int)`
- address of `matrix[i][j]`: address of `matrix + (i * 3 + j) * sizeof(int)`

Declaration of C Arrays

```c
int a[4];                       /* 4 elements, uninitialized */
int b[] = {1, 5, 2, 8};         /* 4 elements */
int c[8] = {1, 5, 2, 8};        /* 8 elements, filled with 0s */
int d[3] = {1, 5, 2, 8};        /* 3 elements, rest is dumped */

extern int e[];
extern int f[10];               /* size ignored */

char s[] = "alma";
char z[] = {'a','l','m','a','\0'};

int m[5][3];                    /* 15 elements, continuously */
int n[][3] = {{1,2,3},{2,3,4}}; /* size cannot be left out! */
int q[3][3][4][3];              /* 108 elements */
```

Indexing of Arrays

C

- `int t[] = {1,2,3,4};`
- indexed from 0
- size is not known at runtime
- during compilation: `sizeof`
  - `sizeof(t) / sizeof(t[0])`
- erroneous index: undefined behaviour

Python

- `t = [1,2,3,4]`
- indexed from 0
- size is known at runtime
- meaning of negative index
  - elements before the last: `t[-2]`
- erroneous index: runtime error

C Pointers

- Can point to other variables: indirection
  - dynamic
  - automatic or static
- Type safe

```
int i;
int t[4];
int *p = NULL;  /* points to nowhere */

/* points to dynamic storage class variable */
p = (int*)malloc( sizeof(int) * i ); ... free(p);

/* points to automatic or static storage class */
p = &i;    p = t;

*p = 5;          /* dereference */
```

C Declarations with Pointers

```c
int i = 42;
int *p = &i;
int **pp = &p;          /* pointer to pointer */
int *ps[10];            /* array of pointers */
int (*pt)[10];          /* pointer to array */

char *str = "Hello!";

void *foo = str;        /* can point to anything */

int* p,q;               /* pointer and int */
int  s,t[5];            /* int and array */
int *f(void);           /* function with int* result */
int (*f)(void);         /* pointer to function with int result */
```

Connection of Arrays and Pointers

- Array: *second-class citizen*
- Array $\rightarrow$ Pointer
- Not equivalent!

```
int t[] = {1,2,3};
t = {1,2,4};  /* compilation error */

int *p = t;
int *q = &t[0];

int (*r)[3] = &t;

printf( "%d%d%d%d\n", t[0], *p, *q, (*r)[0] );
```

Passing Arrays as Parameters?

In real, parameter is a pointer!

```
double distance( double a[], double *b )
{
   double dx = a[0] - b[0],
          dy = a[1] - b[1],
          dz = a[2] - b[2];
   return sqrt(dx*dx + dy*dy + dz*dz);
}
```

Pointer-arithmetic – Stepping

```
int v[] = {6, 2, 8, 7, 3};
int *p = v;
int *q = v + 3;    /* v is converted */
++p;
*p = 5;            /* v: 6, 5, 8, 7, 3 */
p += 2;
*q = 1;            /* v: 6, 5, 8, 1, 3 */
q -= 2;
*q = 2;            /* v: 6, 2, 8, 1, 3 */
```

Pointer-arithmetic – Comparisons

```
int v[] = {6, 2, 8, 7, 3};
int *p = v;
int *q = v + 3;

if ( p == q ) { ... }
if ( p != q ) { ... }
if ( p < q ) { ... }
if ( p <= q ) { ... }
if ( p > q ) { ... }
if ( p >= q ) { ... }
```

Pointer-arithmetic – Indexing

```c
char str[] = "hello";

str[ 1 ] = 'o';
*( str + 1 ) = 'o';

printf( "%s\n", str + 3 );
printf( "%c\n", 3[ str ] );
```

Pointer-arithmetic: Example

```
int strlen( char* s )
{
    char* p = s;
    while( *p != '\0' )
    {
        ++p;
    }
    return p - s;
}
```