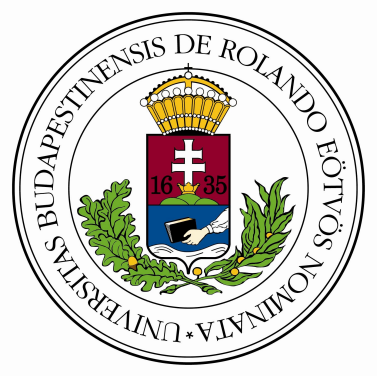
Imperative programming

5th Lecture



Kozsik Tamás

ELTE Eötvös Loránd Tudományegyetem

Outline

Statements we had so far

- Simple Statements
  - Variable Declaration
  - Assignment
  - Subprogram Call
  - Return from Function
- Control Structures
  - Branch
  - Loop

Variable Declaration

C

- Every variable is constructed before first use
- Good to initialize even at this first stage

```
double m;
int n = 3;
char cr = '\r', lf = '\n';
int i = 1, j;
int u, v = 3;
```

Expression-statement in C

Expression (with Side-effect) Evaluation

```
<statement> ::= <expression> ;
              | ...
```

```
n = 1;
x *= y;
c++;
n > 0 ? --n : ++n;   /* not idiomatic! */
```

Typical example: Assignments

Functions in C

- Declared return type, corresponding `return` statement(s)
- Side-effect Only: `void` return value, empty `return`

Pure Function

```
unsigned long fact(int n)
{
  unsigned long result = 1L;
  int i;
  for( i=2; i<=n; ++i )
     result *= i;
  return result;
}
```

Side-effect Only

```
void print_squares(int n)
{
  int i;
  for( i=1; i<=n; ++i ){
     printf("%d\n",i*i);
  }
  return; /* can be left out */
}
```

Mixed Behaviour

```
printf("%d\n", printf("%d\n",42));
```

Return

- Functions can have multiple `return` statements
- No `return` ≡ Empty `return` (in C: `void`)

C

```c
return 42;
return v + 3.14;


return;
```

Multiple return Statements

C

```c
int index_of_1st_negative( int nums[], int length ){
    int i;
    for( i=0; i<length; ++i )
        if( nums[i] < 0 )
            return i;
    return -1;    /* extremal value */
}
```

Empty Statement

C

```
;
```

```
int i = 0;
while( i<10 );
  printf("%d\n",++i);
```

```
int i, nums[] = {3,6,1,45,-1,4};
for( i=0; i<6 && nums[i]<0; ++i);

for( i=0; i<6 && nums[i]<0; ++i){
}
```

Control Structures

- Branch
- Loop
  - Testing - Pre-Testing - Post-Testing
  - Counting
- Non-structured passing of control
  - `return`, `break`, `continue`, `goto`
  - Exceptions, Exception handling

Structured Programming

- Sequence, Branch, Loop
- Every algorithm can be described with these
- More readable, easier to reason for correctness
- Only use something else for a VERY GOOD reason!

Sequence

- Statements, one after the other
- Semicolon: C and Python
- Block Statement in C

```
<statement>      ::= { <statement-list> }
                   | ...

<statement-list> ::= ""
                   | <statement> <statement-list>
```

Internals of Control Structures

C

- One Statement
- Can be one Block Statement

Branch

- if-else structure
  - else-branch is optional
- in C: dangling else

Multi-way Branch

C: idiom

```
if( x > 0 )
    y = x;
else if( y > 0 )
    x = y;
else
    x = y = x * y;
```

Conventional Indentation of Multi-way Branch

C: idiom

```
if( x > 0 )
    y = x;
else if( y > 0 )
    x = y;
else
    x = y = x * y;
```

Conventional Indentation

```
if( x > 0 )
    y = x;
else
    if( y > 0 )
        x = y;
    else
        x = y = x * y;
```

Braces Help

C: idiom

```
if( x > 0 ){
    y = x;
} else if( y > 0 ){
    x = y;
} else {
    x = y = x * y;
}
```

Conventional Indentation

```
if( x > 0 ){
    y = x;
} else {
    if( y > 0 ){
        x = y;
    } else {
        x = y = x * y;
    }
}
```

switch-case-break Statement in C

whole number type, based on compile time constants

```
switch( dayOf(date()) )
{
    case  0: strcpy(name,    "Sunday"); break;
    case  1: strcpy(name,    "Monday"); break;
    case  2: strcpy(name,   "Tuesday"); break;
    case  3: strcpy(name,"Wednesday"); break;
    case  4: strcpy(name, "Thursday"); break;
    case  5: strcpy(name,    "Friday"); break;
    case  6: strcpy(name, "Saturday"); break;
    default: strcpy(name,"illegal value");
}
```

Control Coded in Data

```
char *names[] = {"Sunday", "Monday", "Tuesday", "Wednesday",
                 "Thursday", "Friday", "Saturday"};
strcpy(name, names[n]);
```

Not always convenient

```
switch( key )
{
    case 'i': insertMode(currentRow,currentCol);
              break;
    case 'I': insertMode(currentRow,0);
              break;
    case 'a': insertMode(currentRow,currentCol+1);
              break;
    case 'A': insertMode(currentRow,length(currentRow));
              break;
    case 'o': openNewLine(currentRow+1);
              break;
    case 'O': openNewLine(currentRow);
              break;
}
```

Fall-Through

```
switch( month )
{
    case  1:
    case  3:
    case  5:
    case  7:
    case  8:
    case 10:
    case 12: days = 31;
             break;
    case  2: days = 28 + (isLeapYear(year) ? 1 : 0);
             break;
    default: days = 30;
}
```

Non-trivial Fall-Through

```
switch( getKey() )
{
    case 'q': jump = 1;
    case 'a': moveLeft();
              break;
    case 'e': jump = 1;
    case 's': moveRight();
              break;
    case ' ': openDoor();
}
```

`switch` and Structured Programming

Practically Structured

- `break` at end of all branches
- Same statements for multiple branches

NOT Structured Programming

- Non-trivial Fall-Throughs
- e.g. no `break` at all

Pre-test Loop

C

```c
while( i > 0 )
{
    printf("%i\n", i);
    --i;
}
```

Readability

C

```c
while( i > 0 )
{
    printf("%i\n", i);
    --i;
}
```

C

```c
while( i > 0 )
    printf("%i\n", i--);
```

while – syntax

C

```
<while-stmt> ::= while ( <expression> ) <statement>
```

Post-test Loop

in C

<do-while-stmt> ::= do <statement> while (<expression>);

Typical Example

```
char command[LENGTH];
do {
    read_data(command);
    if( strcmp(command,"START") == 0 ){
        printf("start\n");
    } else if( strcmp(command,"STOP") == 0 ){
        printf("stop\n");
    }
} while( strcmp(command,"QUIT") != 0 );
```

Rewrite – 1

Under what conditions is this true?

`do` σ `while ( ` ε `);` $\equiv$ σ `while ( ` ε `)` σ

Rewrite – 2

Under what conditions is this true?

```
do σ while ( ε );
```

$\equiv$

```
int new_var = 1; ... while ( new_var ){ σ  new_var = ε; }
```

Previous Example Rewritten

```
char command[LENGTH];
int new_var = 1;
...
while( new_var ) {
    read_data(command);
    if( strcmp(command,"START") == 0 ){
        ...
    } else if( strcmp(command,"STOP") == 0 ){
        ...
    }
    new_var = ( strcmp(command,"QUIT") != 0 );
}
```

Refactored

```
char command[LENGTH];
int stay_in_loop = 1;
...
while( stay_in_loop ) {
    read_data(command);
    if(         strcmp(command,"START") == 0 ){
        ...
    } else if( strcmp(command,"STOP" ) == 0 ){
        ...
    } else if( strcmp(command,"QUIT" ) == 0 ){
        stay_in_loop = 0;
    }
}
```

Reading Until End-Of-File Idiom

```
void cat()
{
    int c;
    while( (c = getchar()) != EOF )
    {
        putchar(c);
    }
}
```

Counting Loop

C

```
<for-stmt> ::= for ( <optional-expression> ;
                     <optional-expression> ;
                     <optional-expression> )
                   <statement>
<optional-expression> ::= "" | <expression>
```

(initialization; condition; stepping)

Infinite Loop in C

```
while(1) ...
```

```
for(;;) ...
```

Making the Table of Characters

```
unsigned char c;
for( c = 0; c <= 255; ++c )
{
    printf( "%d\t%c\n", c, c );
}
```

Practical Compilation

```
gcc -ansi -W -Wall -pedantic ...
```

Rewrites

Always works

`while ( ` ε `) ` σ $\quad\Rightarrow\quad$ `for ( ; ` ε `; ) ` σ

Under what conditions is this true?

`for ( ` ι `; ` ε `; ` λ `) ` σ $\quad\Rightarrow\quad$ ι`; while ( ` ε `){ ` σ λ`; }`