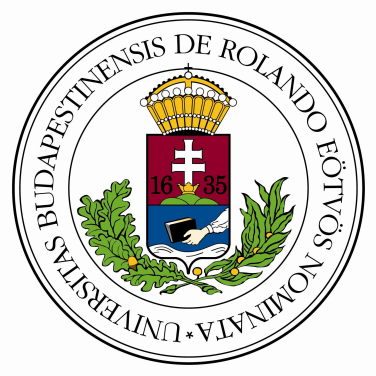
Imperative programming

6th Lecture



Kozsik Tamás

ELTE Eötvös Loránd Tudományegyetem

Outline

Simple Statements

- Variable Declaration Statement
- Empty Statement
- Expression-statement
- Assignment
- Subprogram Call
- Return from Subprogram

Control Structures of Structured Programming

- Block Statement
- Branches
  - `if–elif–else`
  - `switch–case–break`
- Loops
  - Testing Loops
    - Pre-test (`while`)
    - Post-test (`do–while`)
  - Counting (stepping) Loop (`for`)

Non-structured Transfer of Control

- `return`
- `break` and `continue`
- `goto`

break Statement

- Exits the inner loop (or switch)

```
while( !destination(x,y) ){
    drawPosition(x,y);
    dx = read(sensorX);
    if( dx == 0 ){
        dy = read(sensorY);
        if( dy == 0 ) break;
    } else dy = 0;
    x += dx;
    y += dy;
}
```

continue Statement

- Finishes the execution of the inner loop body

```
while not destination(x,y):
    drawPosition(x,y)
    dx = read(sensorX)
    if dx == 0:
        dy = read(sensorY)
        if dy == 0: continue
    else: dy = 0
    if validPosition( x+dx, y+dy ):
        x += dx; y += dy
```

- for-loop executes the stepping as well

goto Statement in C

- Jumps to instruction with designated label inside a function

```
<statement> ::= ...
              | goto <label>
              | <label> : <statement>
<label> ::= <identifier>
```

Search Zero in a Matrix

using goto

```
int matrix[SIZE][SIZE];
...
int found = 0;
int i, j;
for( i=0; i<SIZE; ++i ){
  for( j=0; j<SIZE; ++j ){
    if( matrix[i][j] == 0 ){
      found = 1;
      goto end_of_search;
    }
  }
}
/* --i; --j; */
end_of_search:;
```

neatly

```
int matrix[SIZE][SIZE];
...
  int found = 0;
  int i=-1, j;
  while( i<SIZE-1 && !found ){
    j = -1;
    while( j<SIZE-1 && !found ){
      if( matrix[i+1][j+1] == 0 ){
        found = 1;
      }
      j++;
    }
    i++;
  }
```

Recursive Subprograms

C

```c
int factorial( int n ){
    if( n < 2 ){
        return 1;
    } else {
        return n * factorial(n-1);
    }
}
```

Phrased Differently

C

```c
int factorial( int n )
{
    return n < 2 ? 1 : n * factorial(n-1);
}
```

Repeating Computation Steps

Imperative Programming

- Iteration (Loop)
- Efficient

```
int factorial( int n ){
    int result = 1;
    int i = 1;
    for( i=2; i<=n; ++i )
        result *= i;
    return result;
}
```

Functional Programming

- Recursion
- Easy to understand

```
int factorial( int n ){
    if( n < 2 ) {
        return 1;
    } else {
        return n * factorial(n-1);
    }
}
```

Recursion in Imperative Languages

- Supported in most of the languages
- Rarely used in practice
  - Efficiency
  - Stack overflow

Sometimes it is Convenient

```
int partition( int array[], int lo, int hi );

void quicksort_rec( int array[], int lo, int hi )
{
    if( lo < hi )
    {
        int pivot_pos = partition(array,lo,hi);
        quicksort_rec( array, lo, pivot_pos-1 );
        quicksort_rec( array, pivot_pos+1, hi );
    }
}

void quicksort( int array[], int length )
{
    quicksort_rec(array,0,length-1);
}
```

Tail-recursive Function

- Some functions are already tail-recursive
- Can be rewritten to use tail-recursion (accumulator)

Straightforward

```
int factorial( int n ){
    return n < 2 ? 1 : n * factorial(n-1);
}
```

Tail-recursive

```
int fact_acc(int n, int acc){
    return n < 2 ? acc : fact_acc(n-1,n*acc);
}

int fact( int n ){
    return fact_acc(n,1);
}
```

Compiler Optimizes

Tail-recursive

```
int fact_acc(int n, int acc){
   if (n<2) return acc;
   else return fact_acc(n-1,n*acc);
}
```

Optimized

```
int fact_acc(int n, int acc){
   START: if (n<2) return acc;
   else {
      acc *= n;
      n--;
      goto START;
   }
}
```

Optimized and Structured

```
int fact_acc(int n, int acc){
   while( n>=2 ){

      acc *= n;
      n--;
   }
   return acc;
}
```

Outline

Program Structure

- Program Articulation – logical/physical
- Program Units
  - e.g. subprograms (functions)

Coordinated Structures

- Compilation Units
- Program Libraries
- Reusability

Subordinated Structures

- Nesting
- Hierarchical structure
- Locality: reducing complexity

Hierarchical Construct of Programs

- Nesting Program Units
- Python: Function in Function
  - Block Structured Language
- Reducing Scope: It can only be used where I want.

Without Hierarchy

```
int partition( int array[], int lo, int hi ){ ... }

void quicksort_rec( int array[], int lo, int hi ){
    if( lo < hi ){
        int pivot_pos = partition(array,lo,hi);
        quicksort_rec( array, lo, pivot_pos-1 );
        quicksort_rec( array, pivot_pos+1, hi );
    }
}

void quicksort( int array[], int length ){
    quicksort_rec(array,0,length-1);
}
```

Nesting Functions, Local Definition

Not valid C code!

```
void quicksort( int array[], int length )
{
    int partition( int array[], int lo, int hi ){ ... }

    void quicksort_rec( int array[], int lo, int hi ){
        if( lo < hi ){
            int pivot_pos = partition(array,lo,hi);
            quicksort_rec( array, lo, pivot_pos-1 );
            quicksort_rec( array, pivot_pos+1, hi );
        }
    }

    quicksort_rec(array,0,length-1);
}
```

Declaration and Definition

Often put together, but can also stand independently!

- Declaration: name is given to something
  - Variable Declaration
  - Function Declaration
- Definition: determining what is this something
  - Creation of Variable (Storage Allocation)
  - Function Body Implementation

```
unsigned long int factorial(int n);
int main(){ printf("%ld\n",factorial(20)); return 0; }
unsigned long int factorial(int n){
    return n < 2 ? 1 : n * factorial(n-1);
}
```

Scope of Declaration

While the referred thing is available through the name.

- Global: is outside of everything
- Local: is inside of something

```
unsigned long int factorial( int n )   /* global function */
{
    unsigned long int result = 1L;     /*  local variable  */
    int i;
    for( i=2; i<n; ++i )
    {
        result *= i;
    }
    return result;
}
```

Block

The base of (static) scope rules.

- Subprogram
- Block Statement (C)

Body

Body made of multiple statements:

C

```c
if( lo < hi )
{
  int pivot_pos = partition(array,lo,hi);
  quicksort_rec( array, lo, pivot_pos-1 );
  quicksort_rec( array, pivot_pos+1, hi );
}
```

Static/Lexical Scoping

From declaration to the end of the block containing the declaration.

```
int factorial( int n )
{
    int result = n, i = result-1;  /* cannot be swapped */
    while( i > 1 )
    {
        result *= i;
        --i;
    }
    return result;
}
```

Global – Local Declarations

- Global: no block contains the declaration
- Local: block contains the declaration

Local, Non-local, Global Declarations

- Local to a block: it is in that block
- Non-local to a block:
  - it is in a containing (outer) block
  - but the actual block is in scope of the declaration
- Global: not local to any blocks

Local, Non-local, Global Variable in C

```c
int counter = 0;                                    /* global */
int fun()
{
    int x = 10;                                  /* local to fun */
    while( x > 0 )
    {
        int y = x/2;              /* local to block statement */
        printf("%d\n", 2*y == x ? y : y+1);
        --x;                    /* referring non-local variable */
        ++counter;              /* referring global variable */
    }
}
```

Global Declarations and Definitions in C

```c
int x;

extern int y;

int f(int p);        /* extern can be left out */

int g()
{
    x = f(y);
}
```

Compilation

Every used name should be declared in every compilation unit.

Linking

Every global name should be defined exactly once in the whole program.

Shadowing/Hiding

- Same name is declared on multiple things
- With overlapping (containing) scope

```
void hiding()
{
    int n = 0;
    {
        int n = 1;
        printf("%d",n);
    }
    printf("%d",n);
}
```

- The innermost declaration wins
- Visibility: part of scope

Local Variable Declaration in C

ANSI C

- At beginning of block, before all statements.

from C99

- Mixed with other statements.

```
int n = 0;
{
    printf("%d",n);
    int n = 1;
    printf("%d",n);
}
```

- As a local variable of For-loop

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

Non-local Definitions

```c
int n = 0;
{
    printf("%d",n);
    int n = 1;
    printf("%d",n);
}
```