Imperative programming

Scope

Tamás Kozsik et al

Eötvös Loránd University

October 19, 2022

Table of contents

ELTE
EÖTVÖS LORÁND
TUDOMÁNYEGYETEM

Program structure

- Program segmentation – logical/physical
- Program units

Adjacent structures

- Translation units
- Program libraries
- Reusability

Hierarchical structures

- Nesting
- Hierarchical arrangement
- Locality: reducing complexity

Hierarchical program structure

- Nesting program structures
- Function in function: block structured language
- Reducing scope: it can only be used where I want to use it

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 function 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);
}
```

Nesting functions to any depth?

Not valid C code!

```
void quicksort(int array[], int length)
{
  void quicksort_rec(int array[], int lo, int hi)
  {
    int partition(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 together, but can be one without the other!

- Declaration: name (and type) is given to something
  - Variable declaration
  - Function declaration
- Definition: it's defined what it is
  - Variable creation (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);
}
```

Static/lexical scoping

Scope of declarations

The scope of the declaration is the part of the program where the object referred to by the declaration can be accessed by name.

- The basis of (static) scope rules is the block
    - Sub-program
    - Block statement
- From the declaration to the end of the block directly containing the declaration

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

Global – local declaration

- Global: if the declaration is not contained by a block
- Local: if the declaration is inside of a block

- Local to a block: it's inside of that block
- Non-local to a block
  - The declaration is in an enclosing (outer) block
  - But the current block is in the scope of the declaration
- Global: Not local to any block

Local, non-local, global variables

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

Shadowing/hiding

- Same name declared for multiple things
- With an overlapping (inclusive) scope
- The “inner” declaration wins

```
void hiding()
{
  int n = 0;

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

  printf("%d",n);       /* 0 */
}
```

Local variable declaration

ANSI C

- At the beginning of a block, before other statements

From C99

- Mixed with other statements

```
{
  printf("Hello\n");
  int i = 42;
  printf("World\n");
}
```

- As a local variable of a forloop

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

Order of declarations

Bad

```c
void g() {
  printf("%c", f());
}
char f() { return 'G'; }
```

Good

```c
char f();    /* forward declaration */
void g() {
  printf("%c", f());
}
char f() { return 'G'; }
```

Definition without declaration

```
double x = x + x
six = double 3
zoo = double 10.0
```

```
six = (\x -> x+x) 3
```

```
double = \x -> x+x
six = double 3
```

Higher order functions

Functional programming paradigm

```
filter predicate (x:xs)
  | predicate x = x : filter predicate xs
  | otherwise   = filter predicate xs
filter _ [] = []

filter even [1..10]
filter (\x -> x > 4) [1..10]
filter ( > 4 ) [1..10]
```

Dynapic scoping rules

Bash

```bash
#!/bin/bash
x=1
function g() {
  echo $x;
  x=2;
}
function f() {
  local x=3;
  g;
}

f
echo $x
```

C

```c
#include <stdio.h>
int x = 1;
void g() {
  printf("%d\n", x);
  x = 2;
}
void f() {
  int x = 3;
  g();
}
int main() {
  f();
  printf("%d\n", x);
}
```