COMP26020 Programming Languages and Paradigms -- Part 1

The Main Programming Paradigms

Main Programming Paradigms and Sub-Paradigms

- **Imperative**

    - **Structured** (or procedural)
    - **Object Oriented**
    - **Concurrent**

- **Declarative**:

    - **Functional**
    - **Concurrent**

  Note that there are many other paradigms!

Imperative Programming Paradigm

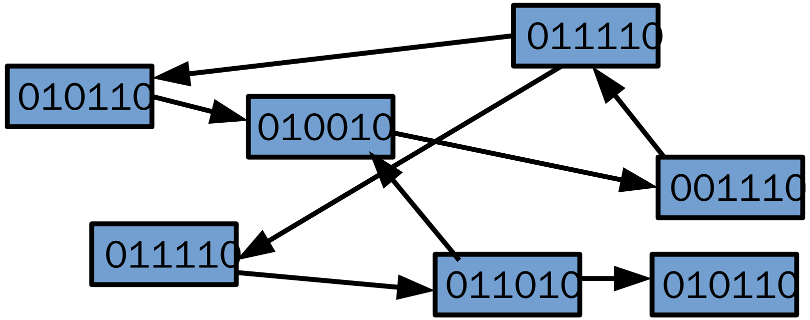
- Programmer describes **sequences of statements** manipulating the program state
  - I.e. describes *how to obtain the computation results*
  - Basically a cooking recipe

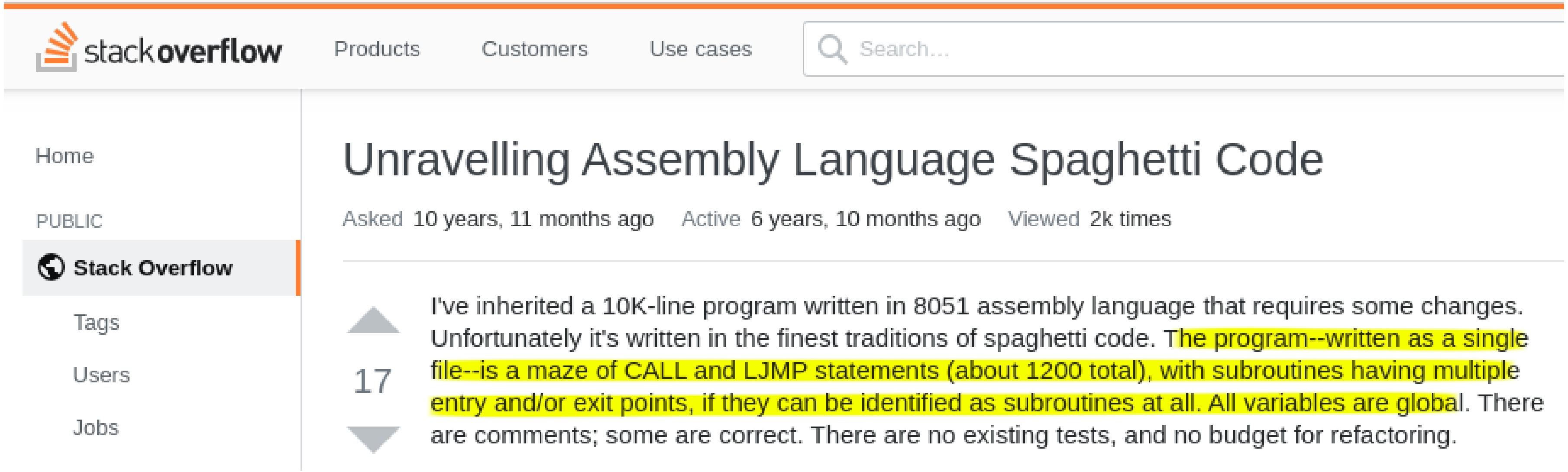
Imperative Programming Paradigm

Example: Intel x86-64 assembly

```
        .global _start
        .text

quit:
        # exit(0)
        mov     $60, %rax                       # system call 60 is exit
        xor     %rdi, %rdi                      # we want return code 0
        syscall                                 # invoke operating system to exit


_start:
        # write(1, message, 14)
        mov     $1, %rax                        # system call 1 is write
        mov     $1, %rdi                        # file handle 1 is stdout
        mov     $message, %rsi                  # address of string to output
        mov     $14, %rdx                       # number of bytes
        syscall                                 # invoke operating system to do the write
        jmp     quit                            # jump to the quit label above

message:
        .ascii  "Hello, world!\n"
```

02-main-programming-paradigms/imperative-asm.S

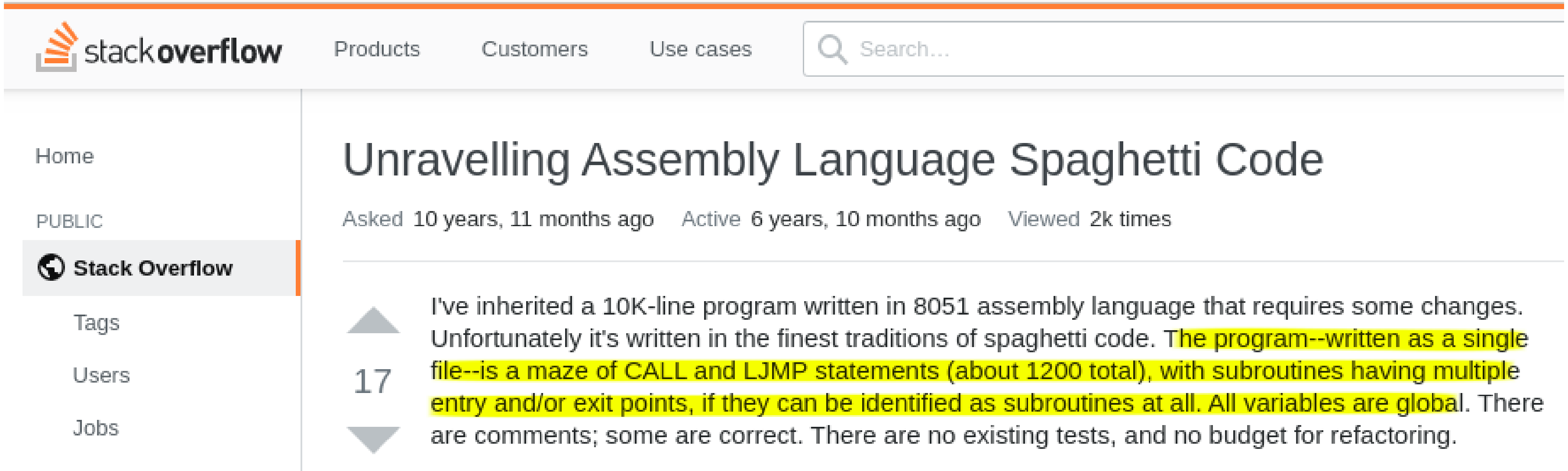

011110
010110
010010
001110
011110
011010
010110



Unravelling Assembly Language Spaghetti Code

Asked 10 years, 11 months ago Active 6 years, 10 months ago Viewed 2k times

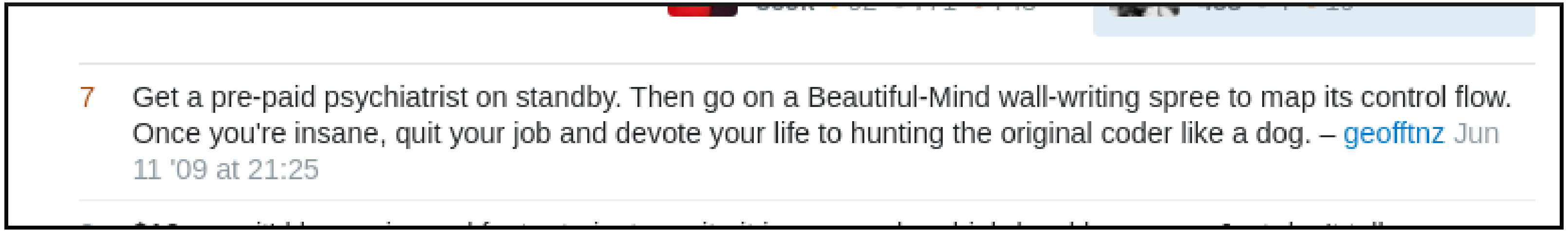
I've inherited a 10K-line program written in 8051 assembly language that requires some changes. Unfortunately it's written in the finest traditions of spaghetti code. The program--written as a single file--is a maze of CALL and LJMP statements (about 1200 total), with subroutines having multiple entry and/or exit points, if they can be identified as subroutines at all. All variables are global. There are comments; some are correct. There are no existing tests, and no budget for refactoring.



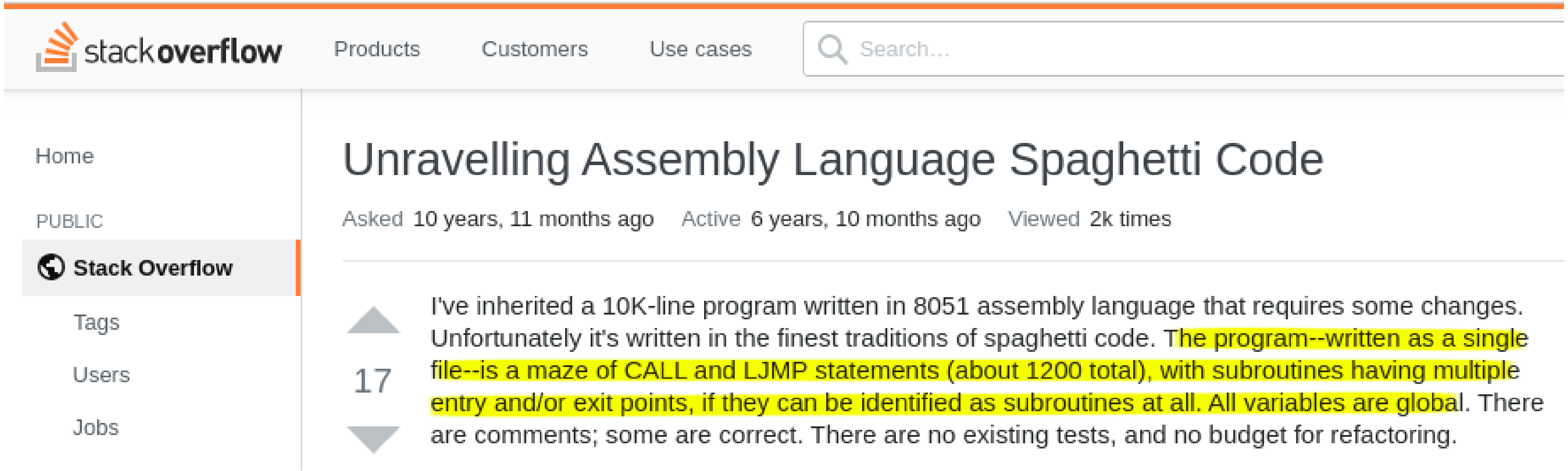
Unravelling Assembly Language Spaghetti Code

Asked 10 years, 11 months ago Active 6 years, 10 months ago Viewed 2k times

I've inherited a 10K-line program written in 8051 assembly language that requires some changes. Unfortunately it's written in the finest traditions of spaghetti code. The program--written as a single file--is a maze of CALL and LJMP statements (about 1200 total), with subroutines having multiple entry and/or exit points, if they can be identified as subroutines at all. All variables are global. There are comments; some are correct. There are no existing tests, and no budget for refactoring.



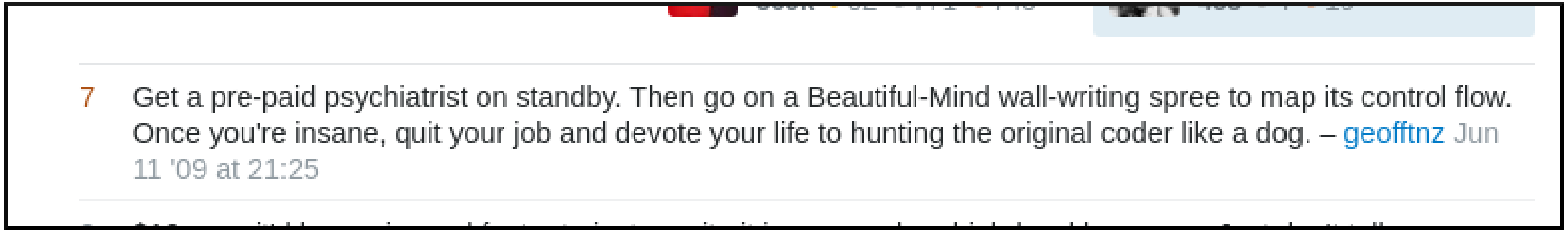
7 Get a pre-paid psychiatrist on standby. Then go on a Beautiful-Mind wall-writing spree to map its control flow. Once you're insane, quit your job and devote your life to hunting the original coder like a dog. – geofftnz Jun 11 '09 at 21:25



Unravelling Assembly Language Spaghetti Code

Asked 10 years, 11 months ago Active 6 years, 10 months ago Viewed 2k times

I've inherited a 10K-line program written in 8051 assembly language that requires some changes. Unfortunately it's written in the finest traditions of spaghetti code. The program--written as a single file--is a maze of CALL and LJMP statements (about 1200 total), with subroutines having multiple entry and/or exit points, if they can be identified as subroutines at all. All variables are global. There are comments; some are correct. There are no existing tests, and no budget for refactoring.

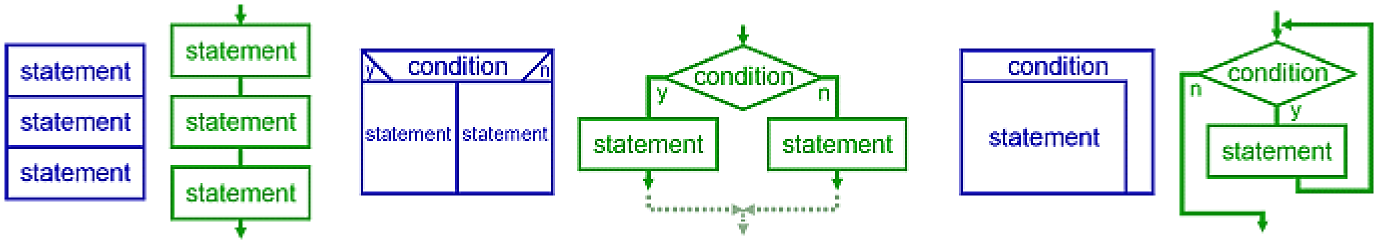


7 Get a pre-paid psychiatrist on standby. Then go on a Beautiful-Mind wall-writing spree to map its control flow. Once you're insane, quit your job and devote your life to hunting the original coder like a dog. – geofftnz Jun 11 '09 at 21:25

Other examples of unstructured imperative languages: early versions of FORTRAN, COBOL, BASIC

Imperative Structured Programming Paradigm

- The programmer uses advanced **control flow operations**
  - Loops, conditionals, procedures
  - Compared to pure imperative, easier to describe/reason about complex/large programs



Imperative Structured Programming Paradigm

Example: C

```c
/* Check if a number is prime */
int is_prime_number(int number) {
    if(number < 2)
        return 0;

    for(int j=2; j<number; j++)
        if(number % j == 0)
            return 0;

    return 1;
}

int main(void) { /* Check which of the first 10 natural integers are prime */
    int i;
    int total_iterations = 10;

    for(i=0; i<total_iterations; i++)
        if(is_prime_number(i))
            printf("%d is a prime number\n", i);
        else
            printf("%d is not a prime number\n", i);

    return 0;
}
```

02-main-programming-paradigms/imperative-structured.c

Imperative Structured Programming Paradigm

Algol

PASCAL

Ada

FORTRAN

Imperative Object-Oriented Programming Paradigm

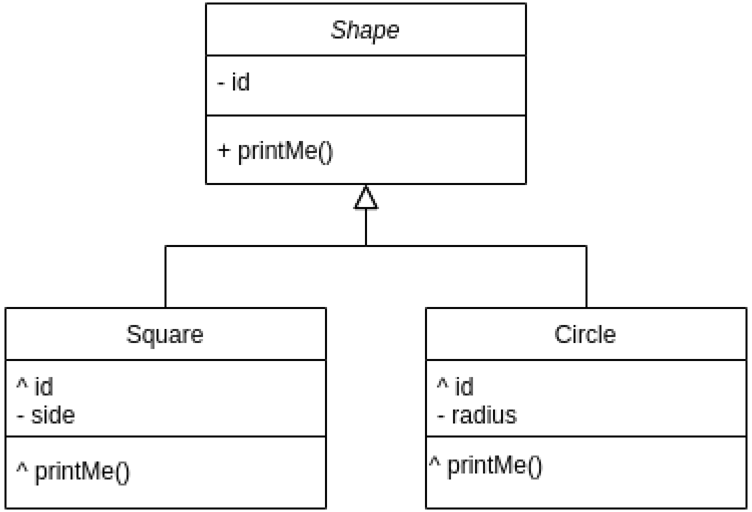
- **Encapsulation** of code and the associated data into objects

Non-OO (procedural):

```
operation(data)
```

OO:

```
data.operation()
```



Imperative Object-Oriented Programming

```
// Example in C#
abstract class Shape {
    public abstract void printMe();
    /* ... */
}

class Square : Shape {
    override void printMe() {Console.WriteLine ("Square id: {0}, side: {1}", _id, _side);}
    /* ... */
}

class Cicle : Shape {
    override void printMe() {Console.WriteLine ("Circle id: {0}, radius: {1}", _id, _radius);}

}

public class MainClass {
    public static void Main(string[] args) {
            Square mySquare = new Square(42, 10);
            Circle myCircle = new Circle(242, 12);
            mySquare.printMe();
            myCircle.printMe();
    }
}
```

02-main-programming-paradigms/imperative-oo.cs

Imperative Object-Oriented Programming

```cpp
// Example in C++
class Shape {
public:
    virtual void printMe(void) = 0;
    /* ... */
};

class Square : Shape {
public:
    void printMe(void) { cout << "Square id: " << _id << ", side: " << _side << endl; }
    /* ... */
};

class Circle : Shape {
public:
    void printMe(void) { cout << "Circle id: " << _id << ", radius: " << _radius << endl; }
    /* ... */
};

int main(void) {
    Square mySquare = Square(42, 10);
    Circle myCircle = Circle(242, 12);
    mySquare.printMe();
    myCircle.printMe();
}
```

[02-main-programming-paradigms/imperative-oo.cpp]()

Imperative Object-Oriented Programming Paradigm

- Well suited to represent problems with a lot of state/operations
  - GUI, simulators, video games, business management software, and many other use cases
- Ease reasoning about / organising large codebases
- Can sometimes be overkill for small programs

Imperative Concurrent Programming Paradigm

- The programmer uses execution threads/processes to describe interleaving and/or parallel computation flows

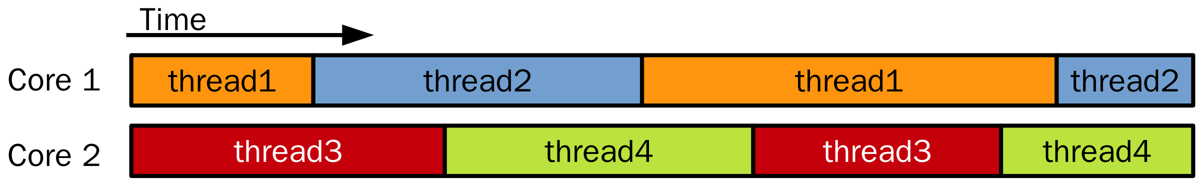
Example in C with POSIX threads:

```c
static void *thread_function(void *argument) {
    int id = *(int *)argument;

    for(int i=0; i<10; i++)
        printf("Thread %d running on core %d\n", id, sched_getcpu());
}

int main(void) {
    pthread_t threads[NUMBER_OF_THREADS];
    int thread_ids[NUMBER_OF_THREADS];

    for(int i=0; i<NUMBER_OF_THREADS; i++) {
        thread_ids[i] = i;
        pthread_create(&threads[i], NULL, &thread_function, &thread_ids[i]);
    }

    /* ... */
```

02-main-programming-paradigms/pthread.c

Imperative Concurrent Programming Paradigm

Execution example with 4 threads and 2 parallel processing units (cores):



- Many imperative languages provide ways to exploit concurrency:
  - Shared-memory threads/processes in C/C++, Java, Python, etc.
  - Message passing (for example MPI) in C/C++/FORTRAN
  - Semi-automatic loop paralelization with libraries such as OpenMP
  - GPU programming with CUDA
  - ...
- Use cases: HPC, distributed computing, graphic processing, etc.

Declarative Programming Paradigm

The programmer describes the **meaning/result of computations**

```
<html>
    <!-- ... --!>
    <body>

        <h1> Hello, world! </h1>
        <p>
            Lorem ipsum dolor sit amet, consectetur adipiscing elit. Vestibulum
            sed leo sit amet urna accumsan aliquam. Fusce et aliquet nibh.
        </p>
    </body>
</html>
```

See the result here. These slides are also created with a combination of 2 declarative languages: HTML and Markdown -- try typing ctrl+U :)

Declarative Programming Paradigm

- High level of abstraction
- Code can easily become convoluted
- Languages: HTML, SQL, XML, CSS, Latex (non-Turing complete)
- Usage: document rendering, structured data storage and manipulation

Declarative Functional Programming Paradigm

- Calling and composing functions to describes the program

Example in Haskell:

```haskell
add_10 x = x + 10

twice f = f . f

main = do
    print $ twice (add_10) 7
```

02-main-programming-paradigms/functional.hs

Declarative Functional Programming Paradigm

```
(* Example in OCaml *)
let width, height = 800, 600
let pi = 4. *. atan 1.;;

let endpoint x y angle length =
    x +. length *. cos angle,
    y +. length *. sin angle;;

let drawLine x y angle length width =
    let x_end, y_end = endpoint x y angle length in
        set_line_width (truncate width);
        moveto (truncate x) (truncate y);
        lineto (truncate x_end) (truncate y_end);;

let rec drawRec x y angle length width =
    if length > 0. then
        let endx, endy = endpoint x y angle length in
            drawLine x y angle length width;
            drawRec endx endy (angle +. pi *. 0.133) (length -. 4.) (width *. 0.75);
            drawRec endx endy (angle +. pi *. -0.166) (length -. 4.) (width *. 0.75);;

moveto 400 200;;
drawRec 400. 200. (pi *. 0.5) 50.0 4.;;
```

02-main-programming-paradigms/functional.ml

Source: https://github.com/DaQuirm/ocaml-fractals

Declarative Functional Programming Paradigm

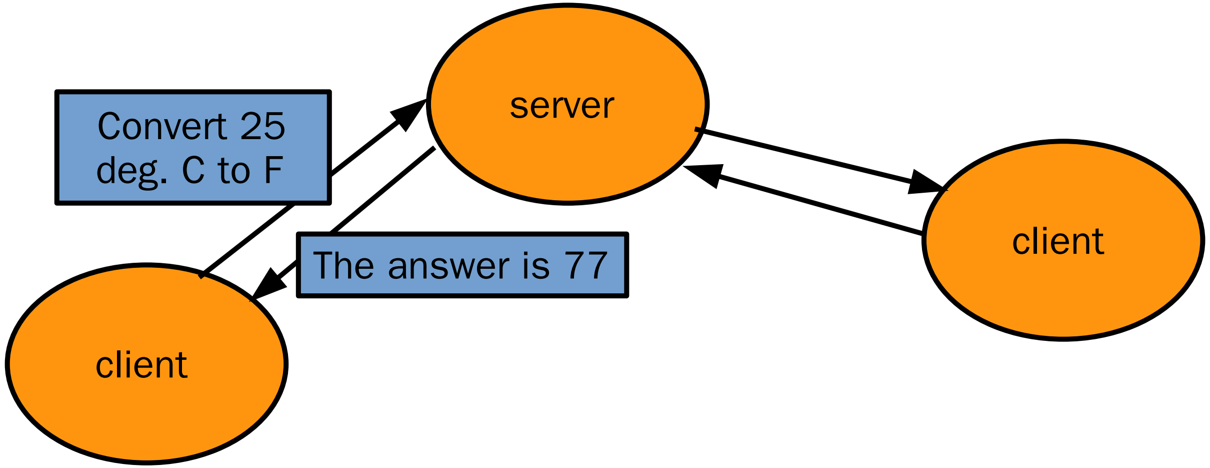
- First-class/higher-order functions
- Loops implemented with **recursion**
- **Pure functions have no side-effects**
- Languages: Haskell, Scala, F#, etc.

Declarative Concurrent Programming Paradigm

```erlang
% Example in Erlang using the actor model

server() ->
    receive
        {From, {convert, TempC}} -> From ! {converted, 32 + TempC *9/5},
                            server();
        {stop} -> io:format("Stopping~n");
        Other -> io:format("Unknown: ~p~n", [Other]),
                 server()
    end.

client(ClientID, ServerPID) ->
    TempC = rand:uniform(40),
    ServerPID ! {self(), {convert, TempC}},
    receive
        {converted, TempF} -> io:fwrite("~p: ~p deg. C is ~p deg. F~n.",
                                [ClientID, TempC, TempF]),
                           timer:sleep(100),
                           client(ServerPID);
        {stop} -> io:format("Stopping~n");
        Other -> io:format("Unknown: ~p~n", [Other])
    end.

start() ->
    Pid1 = spawn(temperature, server, []),
    spawn(temperature, client, [0, Pid1]),
    spawn(temperature, client, [1, Pid1]),
```

02-main-programming-paradigms/declarative-concurrent.erl

Declarative Concurrent Programming Paradigm



- Less need for synchronisation
- Use cases: distributed applications, web services, etc.

There are Many Other Programming Paradigms

- **Logic**, **Dataflow**, **Metaprogramming/Reflexive**, Constraint, Aspect-oriented, Quantum, etc.

https://en.wikipedia.org/wiki/Template:Programming_paradigms

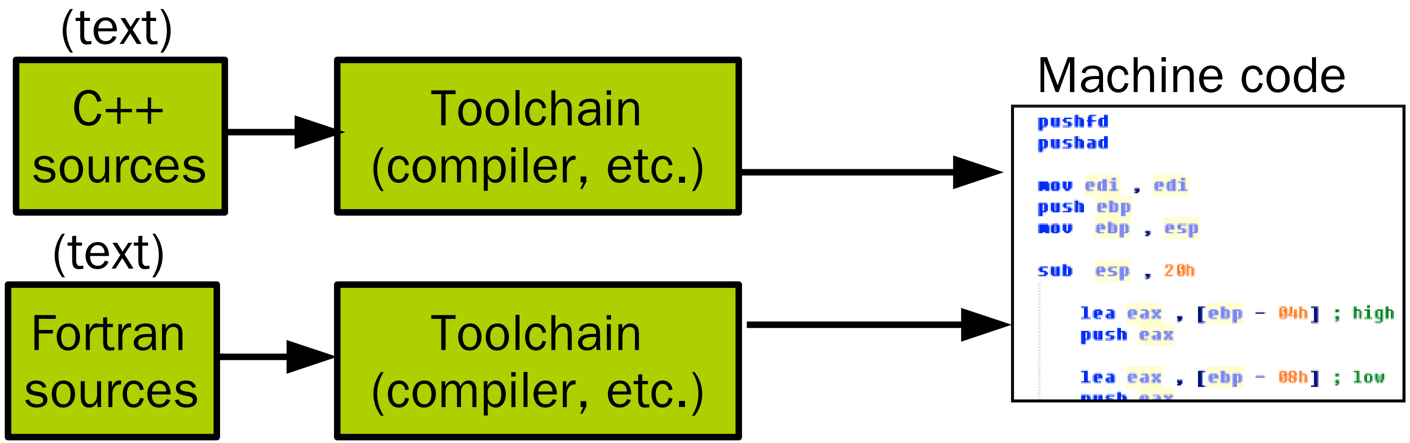
Multi-Paradigm Languages

- Haskell: purely functional and does not allow OO style
- OCaml: mainly functional, allows OO and imperative constructs
- C, C++: imperative but allow some functional idioms:

```
int fact(int x) {

    if(x == 0) return 1;
    return x * fact(x-1);
}
```

- So many languages are **multi-paradigm**

It all Boils down to Machine Code



(text)

C++ sources

Toolchain (compiler, etc.)

(text)

Fortran sources

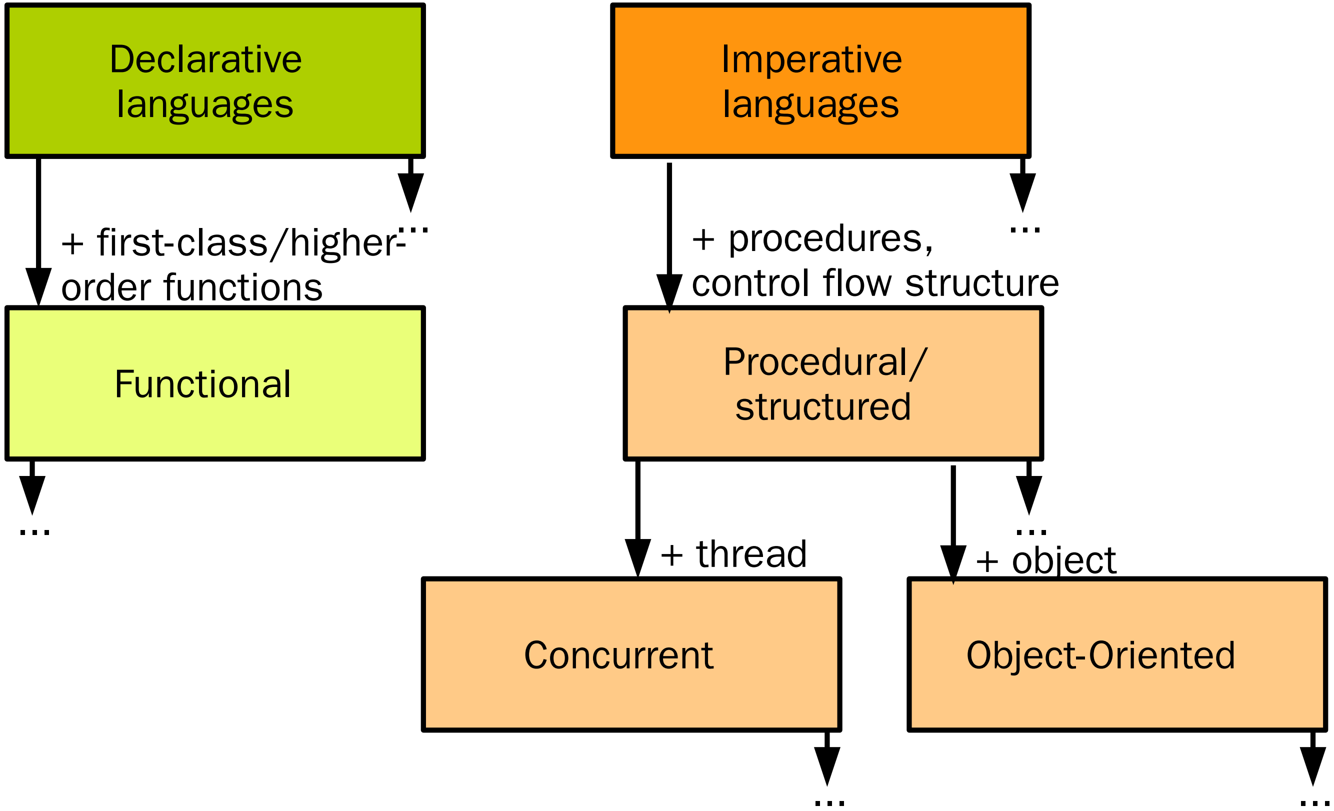
Toolchain (compiler, etc.)

Machine code

```
pushfd
pushad

mov edi , edi
push ebp
mov  ebp , esp

sub  esp , 20h

    lea eax , [ebp - 04h] ; high
    push eax

    lea eax , [ebp - 08h] ; low
    push eax
```

Summary

Inspired from
https://www.info.ucl.ac.be/~pvr/VanRoyChapter.pdf



Declarative languages

+ first-class/higher-order functions

Functional

…

…

Imperative languages

+ procedures, control flow structure

Procedural/ structured

…

…

+ thread

Concurrent

…

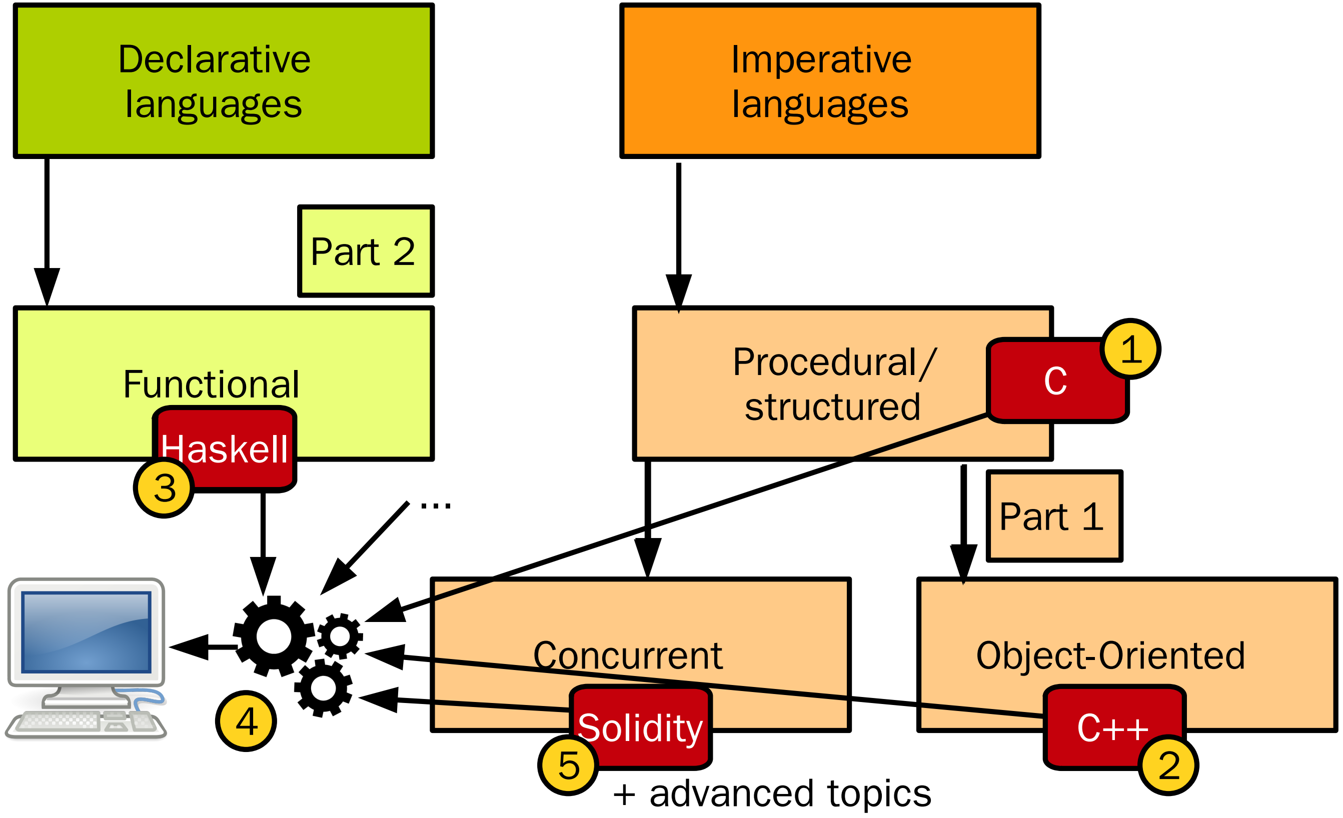
+ object

Object-Oriented

…

Course Overview

Inspired from
https://www.info.ucl.ac.be/~pvr/VanRoyChapter.pdf



Feedback Form

https://bit.ly/3lJIvWq