ENGR101: Lecture 7

Program design

March 2024

What we cover?

- Program design
- Flow Chart
- Pseudo-code
- Program graph

Pseudo-code

When you start new program you don't care about programming language or syntax. Good practice is to get your **logic** first and structure your code accordingly.
Half of the hour spent on design of your software will save you hours later.
If your code is a unstructured mess - it takes ages to make it work.
So you write program's using **pseudo-code** - looks a bit like a code but can be read by somebody who does not know this specific programming language, so avoid language-specific keywords.
Write the whole thing before starting to code.

Constructs of Pseudo-code

There is no fixed standard for pseudocode (attempts to create software which converts customer requirements, written in high-level pseudocode are going, so far without success).
For no particular reason, we picked following:

1. variable - make variable intention clear
   Append the file extension to the name (good)
   name = name + ext (bad)
2. set - not all languages use =
3. if-then-else
4. for
5. function call

Major rule for pseudo-code is : **Should be readable by somebody who does not know this particular language**

So instead of...

```
int maximum(std::vector<int> v){
    int max_val = v.at(0);
    for (unsigned int i = 1 ; i < v.size(); i++){
        if ( v.at(i) >= max_val ) {
            max_val = v.at(i);
        }
    }
    return max_val;
}
```

or same in Python...

```
def find_max(arr):
    max_val = arr[0]
    for num in arr:
        if num > to_max_val:
            max_val = num
    print(max_val)
    return max_val
```

you write:

1. **FUNCTION** get-maximum
    1. Set "maxValue" to the first value in the array.
    2. **For** each element in the array, starting with the second one:
        1. **IF** the current value is greater than "maxValue", set "maxValue" to the current value.
        2. **END IF**
    3. **END FOR**
    4. **RETURN** "maxValue".
2. **END FUNCTION**

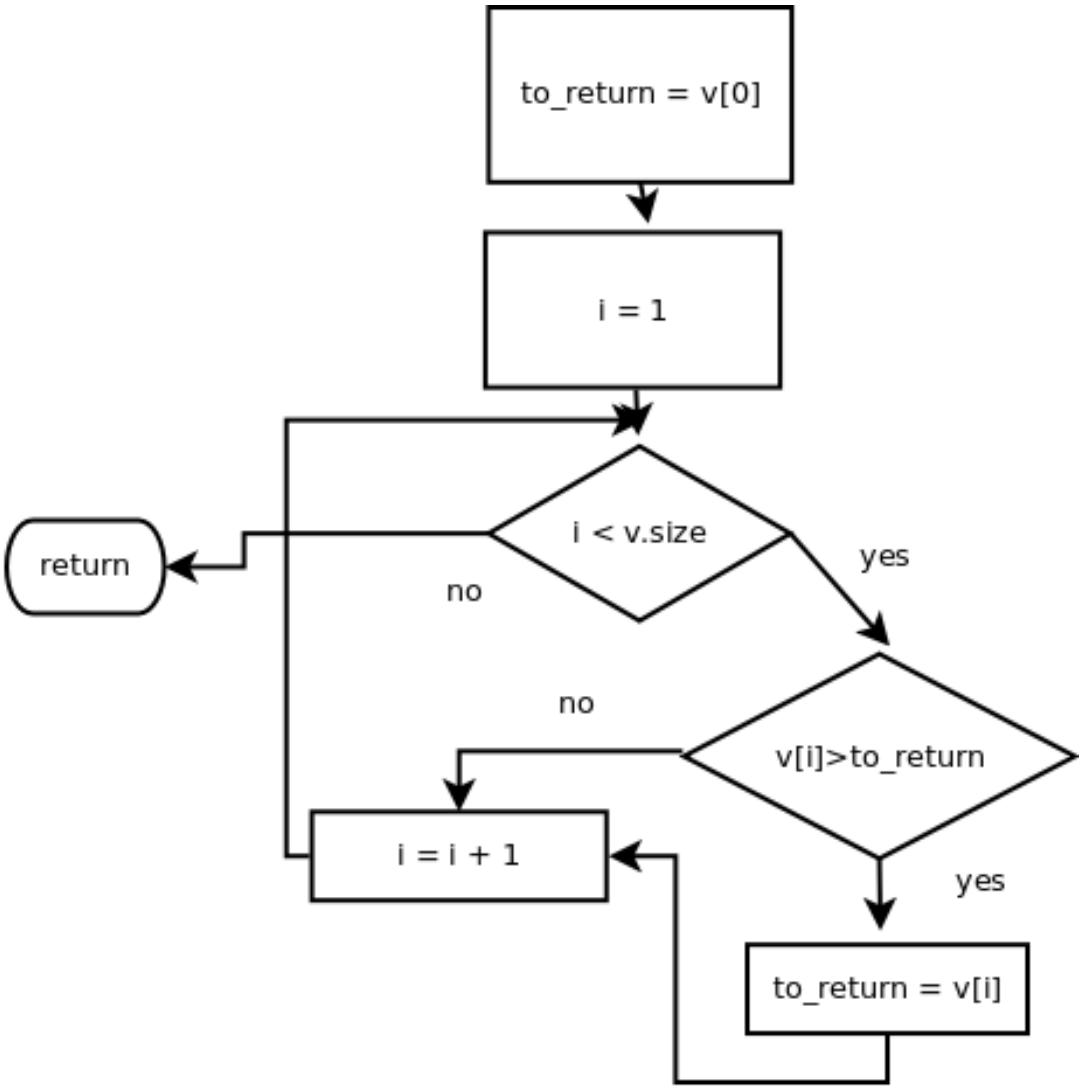
1. Write only one statement per line.
2. Write what you mean, not how to program it
3. Give proper indentation to show hierarchy and make code understandable.
4. Make the program as simple as possible.
5. Conditions and loops must be specified well, i.e. begun and ended explicitly.

Flow Charts

Graphical Representation of program flow.

1. simple calculation is rectangle
2. branching - diamond shape
3. end/return from function - rectangle with rounded corners

Flow Charts



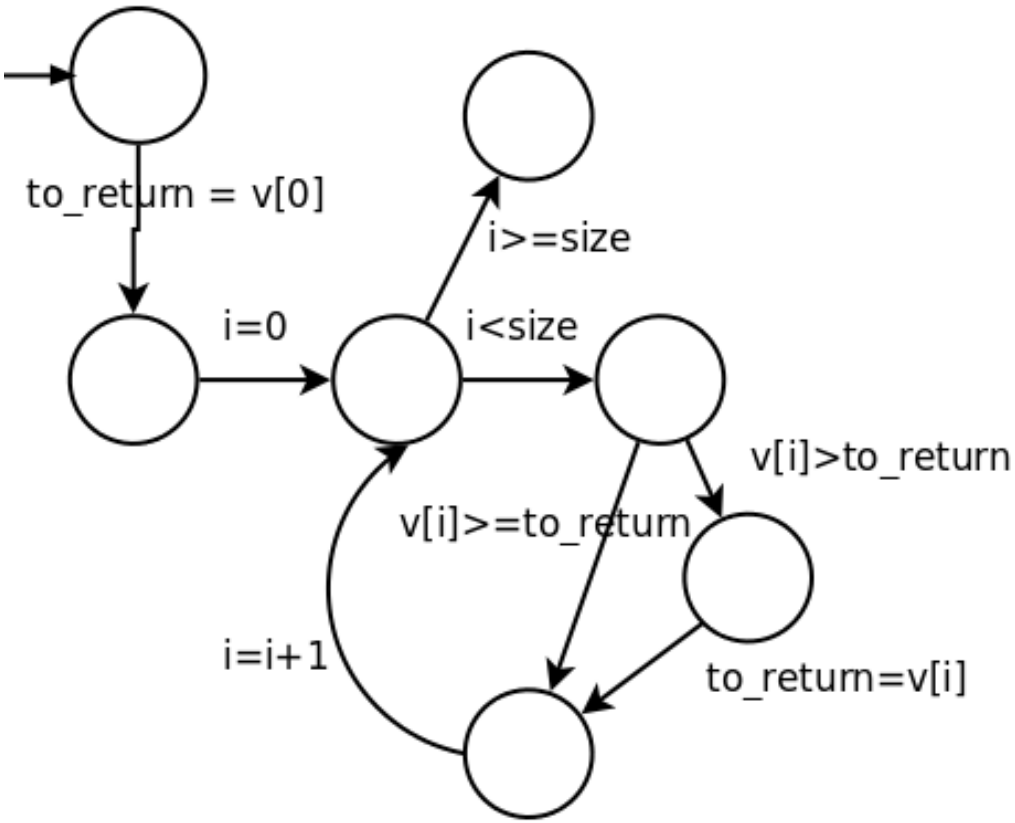

```
1  int maximum(std::vector<int> v){
2      int to_return = v.at(0);
3      for (unsigned int i = 1 ; i < v.size(); i++){
4          if ( v.at(i) >= to_return ) {
5              to_return = v.at(i);
6          }
7      }
8      return to_return;
9  }
```

Program Graphs

Flow Chart are usually used for not very technical presentations.
Program Graphs, while somewhat similar, are more technical.
Program Graph:

- Program can be in one of several **states**.
- Program can move from one state to another one.

Program Graphs




```
int maximum(std::vector<int> v){
    int to_return = v.at(0);
    for (unsigned int i = 1 ; i < v.size(); i++){
        if ( v.at(i) >= to_return ) {
            to_return = v.at(i);
        }
    }
    return to_return;
}
```

Loops are represented by adding an arrow back to the statement where the loop started from.
In a for loop, the arrow is added back from the loop exit condition, in the example above, loop exit condition would be i ¡ y, so an arrow is added back from there to the beginning of the loop. In the example, the exit condition is considered as statement 4.