APS 105 Lecture 15 Notes

Last lecture: Example larger program - Goldbach conjecture

Today: Scope and Arrays of Data

Scope of variables

The set of C statements where a variable is defined / visible / usable

✓ Variables inside functions are only scoped within functions
- local variables / internal identifiers

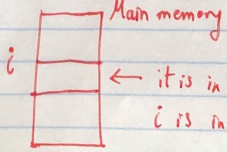
e.g.
```
int func (int x){        ↑ scope          int main (){
                         | of                  int x;   ↑ scope of x
     int y=2;  ↕ scope   | x                            ↓
}                of y    ↓                }
```

✓ You need to declare a variable before using it

~~i=1;~~ compile-time
~~int i;~~ error

Note: initialization → declaring a variable without initialization, means it holds a "garbage" value

e.g.
```
int i;
i = i+1;
```



← it is incorrect to assume i is initialized to 0.

Another Note: initializing a pointer with NULL

e.g. `int *p;`

p will have garbage address, if you try to access an undetermined address it may (or may not) give you an error.

`*p = 6;` → This line may work, may get me bus error or segmentation fault.

If you set p to NULL & do `*p=6` you'll get segmentation fault - which is good news as it is a deterministic error. You will then know that you need to make p point to a valid address

✓ Variables declared within compound statements are only available within the statement.

```
e.g. {
        int x = 2;   ↕ scope of x
     }
```

Note: in another scope, you can re-use variable name, but not recommended as it is very error-prone

✓ External identifiers/global variables: variables declared at the top of your program .c file → these are scoped to all functions → error prone too, avoid using it

e.g.

```
#include <stdio.h>
int x;

void swap(){
    =

}
int main(){

    =

}
```

scope of x

Example:

```
int foo (int x) {

    int y = 5;
    for(int i=0; i<10; i++){
        int z =3;
        y = z + x;
    }
    return y;
}
```

scope of z

scope of i

Scope of y

Scope of x

Scanned with CamScanner

Overlapping Scope: Avoid overlapping scope & avoid reusing variable names

```
int i=1;
printf("i=%d \n", i);
{
    int i=2;   <- new i within scope of another i
    printf("i= %d\n", i);
}
printf("i=%d \n", i);
```

scope of new i

scope of old i

Output :

```
i=1
i=2
i=1
```

Arrays:

So far each identifier /variable name is for 1 value

It is more powerful to deal with many numbers/values under same name → less software written, more work done by computer

e.g.

Assign #	1	2	3	4	5
lines of Code	40	120	450	350	90

To calculate the average, is it optimum to have variables linesCode 1, lines Code 2 - - - - . ?

Instead we can have $lineCode_1$, $lineCode_2$ - as we do in algebra

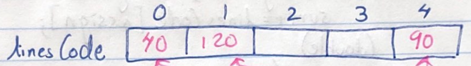
Scanned with CamScanner

Declare an array which can have more than 1 value

e.g.
<u>Declaration:</u>

int linesCode [5]; ← creates 5 variables

① linesCode [0] ← We always start from 0
② linesCode [1]
③ linesCode [2]
④ linesCode [3]
⑤ linesCode [4]



<u>Initialization:</u>

linesCode [0] = 40;
linesCode [1] = 120;
⋮
linesCode [4] = 90;

OR Declaration + Initialization: "you may/may not specify size"

int linesCodes [5] = { 40, 120, 450, 350, 90 };
OR
int linesCode [ ] = { 40, 120, 450, 350, 90 };

Another handy feature:
Recall const int Pi = 3.14;

We can also say

#define $\underbrace{\ }$ PI $\underbrace{\ }$ 3.14
space

→ gets substituted in code:
→ not a variable, no type
It is a macro

Example program that calculates the average lines of code you have written in your assignments:

```
#include <stdio.h>
#define NUM_ASSIGN 5

        int linesCode[] = {40, 120, 450, 350, 90};
        int sum = 0;
        double avg = 0;

        for (int assign = 0; assign < NUM_ASSIGN; assign++)
                sum += linesCode[assign];
              (double)
        avg = ^sum / NUM_ASSIGN;

        printf("Average lines of code is %.2lf \n", avg);
        return 0;
}
```

Scanned with CamScanner