APS105 Lecture 14 Notes

Last lecture: More pointers, passing & returning pointers
Today: scope and Goldbach conjecture

Scope of a variable: the set of C statements where a variable is defined/visible/usable

✓ Variables inside functions are only scoped within functions – local variables

e.g.

```
int func(int x){
    int y = 2;
}
```

scope of y

scope of x

✓ Declare a variable before using it

~~i = 1;~~
~~int i;~~

compiler error

✓ Variables declared within compound statements are only available within that statement

```
{
    int x = 2;
}
```

scope of x

variable names can be reused, but don't
→ very error prone

✓ External identifiers, variables declared at the top of program .c file and are scoped/visible/available to all functions
- called global variable

avoid using it as it is error-prone

e.g.

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

void swap () {
    ==

}
int main (void) {
    =
}
```

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

i

y

x

Overlapping Scope:

```
int i=1;
printf("i= %d \n", i);
{
    int i= 2;          ← defining new i within scope
    printf("i = %d \n", i);
}
printf("i= %d \n", i);
```

scope of i

scope of new i

Output?

i= 1
i= 2
i= 1

Avoid reusing variable names
Avoid overlapping scope

Write a program in C that tests Goldbach conjecture

"Every even integer > 2 can be expressed as a sum of 2 prime numbers"

Prime number : # divisible by 1 and itself. e.g. 2, 3, 5, 7, 11, 13, 17

Goal: program that asks for even # and checks if the conjecture is true or false.

Main steps:

1] Get input from user

2] Check if it is even and greater than 2, if not prompt user again

3] Check the conjecture. How? Think of toy example. e.g. user input is 12

* need a function that generates prime numbers

* Start with 1st prime # 2, 2nd # is 12 - 2 = 10, is 10 prime? NO.

* Go to the next prime # 3, 2nd # is 12 - 3 = 9, is 9 prime? NO.

and so on

Prime #	12 - Prime #	Result
2	10	10 not prime
3	9	9 not prime
5	7	7 is prime - conjecture verified
7	5	

is this step necessary?

→ When should we stop looking for next prime #?

We should stop beyond 12/2, since after that 1st # will be same as 12 - 1st #, or when Prime # > 12 - Prime #

Let's write pseudo-code for step 3, that takes integer N > 2 and checks conjecture by looking for 2 primes FirstPart + SecondPart = N and returns true if found.

```
firstPart = 2
while (conjecture not verified and not rejected){
    secondPart = N - firstPart
    if (secondPart < firstPart)
        conjecture rejected!
    else if (secondPart is prime)
        conjecture verified
    else
        firstPart = next prime Number
}
```

Convert pseudo-code to C code

```
bool testGoldbach(int N){
    int firstPart = 2;
    bool stop = false;
    bool verified = false;
    int secondPart;

    while(!stop){
        secondPart = N - firstPart;

        if (secondPart < firstPart){
            stop = true;
        }
        else if (isPrime(secondPart)){
            stop = true
            verified = true;
        }
        else
            NextPrime(&firstPart);
    }

    return verified;
}
```

will need to implement

need to implement

How to write NextPrime?

→ look at next number (+1) and check if its prime

repeat NOT

```
void NextPrime (int *p){
    int value = *p+1;
    while(!isPrime(value)){ → exist loop when value is prime.
        value += 1;
    }
    *p = value;
}
```

How to write isPrime?

Check that the number is only divisible by itself and 1 → loop over numbers from 2 to number-1

```
bool isPrime (int num){
    bool prime = true;
    if (num < 2)
        prime = false;
    else {
        for (int j = 2; j < num && prime; j++){
            if (num % j == 0)
                prime = false
        }
    } return prime;
```