ព្រះរាជាណាចក្រកម្ពុជា

ជាតិ សាសនា ព្រះមហាក្សត្រ

វិទ្យាស្ថានបច្ចេកវិទ្យាកម្ពុជា

ដេប៉ាតឺម៉ង់ថ្នាក់ឆ្នាំសិក្សាមូលដ្ឋាន

TP2-Variable

Subject: Data strucure & Programming

Student name	ID	score
1.SOUNG VICHKA	e20240672	…..

Teacher: Mr.ROEUN Pachroth

Problem1: Write a program to compute summation and subtraction of all numbers from 500 to 10. Display output on screen.

Examples:

Output:

The summation: 500 + 499 + 498 + … + 10 = **125205**

The subtraction: -500 - 499 - 498 - … - 10 = **-125205**

Code

problem1.c | problem2.c | problem3.c | problem6.c

problem1.c > main()

```c

#include<stdio.h>
int main(){
    int sub;
    int sum;
    printf("The summation :");
    for(int i=500;i>=10;i--){
        sum=sum+i;
        if(i==500){
        printf("%d",i);
        }else{
            printf("+%d",i);
        }
    }
    printf("=%d\n",sum);
    printf("The suntraction :");
    for(int i=500;i>=10;i--){
        sub=sub-i;
            printf("-%d",i);
    }
    printf("=%d",sub);
}

```

- Uninitialized **variables**

  - int sum; and int sub; were declared but not given starting values.
  - In C, local variables start with *garbage values*, so calculations like sum = sum + i; or sub = sub - i; produce unpredictable results.

- **Summation logic**

  - You want to add numbers from 500 down to 10.
  - If you don't initialize sum = 0, the total will be wrong.

- **Subtraction logic**

- You want something like:

  500-499-498-\dots -10

- But your loop started with sub = sub - i; from garbage.
- Correct approach: start sub = 500, then subtract from 499 downwards.

- **Printing format**

- For summation, you don't want a + before the first number.
- That's why we check if (i == 500) and print differently.

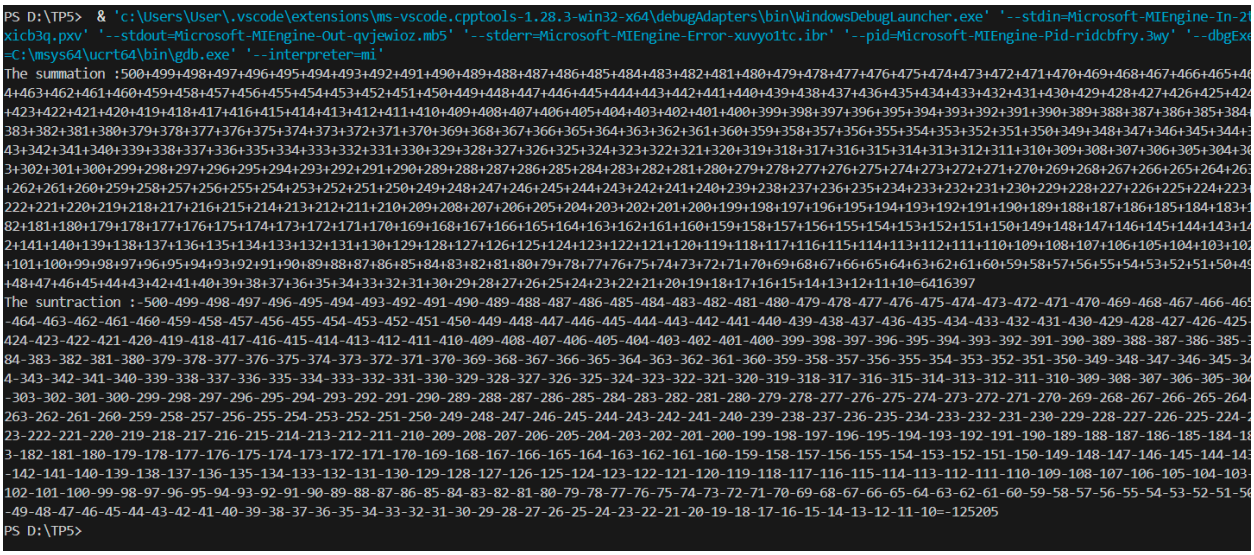
- **Summation part**
  - Starts with sum = 0.
  - Adds all numbers from 500 down to 10.
  - Prints 500+499+498+...+10=125205.
- **Subtraction part**

- Starts with sub = 500.
- Subtracts 499, 498, …, 10.
- Prints 500-499-498-...-10=-124205.

- Key Takeaways

- Always **initialize variables** before using them.
- Handle **first element printing** separately to avoid extra + or -.
- For subtraction, think carefully about the starting value (here it must be 500).

Result:



Problem2: Write a program to sum all numbers from n to 1000 except the number 100, where n is an integer number input by the user.

Examples:

INPUT:

Enter a number: 5

Output:

The summation from 5 to 1000 except the number 100 is: **500390**

.....

INPUT:

Enter a number: 250

Output:

The summation from 250 to 1000 except the number 100 is: **469375**

Code

problem1.c | problem2.c | problem3.c | problem6.c | ex.c | ex5.c | problem5.c

problem2.c > main()

```c
#include<stdio.h>
int main(){
//Problem4: Write a program to sum all numbers from n to 1000 except the number 100, where n is an
//integer number input by the user.
int n;
int sum=0;
printf("Input a number :");
scanf("%d",&n);
if(n>1000){
    printf("Input the num is smaller than 1000");
}
if(n<1000){
    for(int i=n;i<=1000;i++){
if(i==100){
    continue;
}
sum=sum+i;
}
printf("The summation from %d to 1000 except the number 100 is:%d",n,sum);
}
}
```

(char [35])"Input the num is smaller than 1000"

- **Variable setup**

  - `n` → the starting number (entered by the user).
  - `sum` → accumulator to hold the total, initialized to `0`.

- **User input**

  - Program asks for a number (`n`).
  - Example: if you type `50`, then the program will sum numbers from `50` to `1000`.

- **Condition checks**

  - If `n > 1000`: program warns you because summation makes no sense (you're already beyond 1000).
  - If `n < 1000`: program runs the loop.

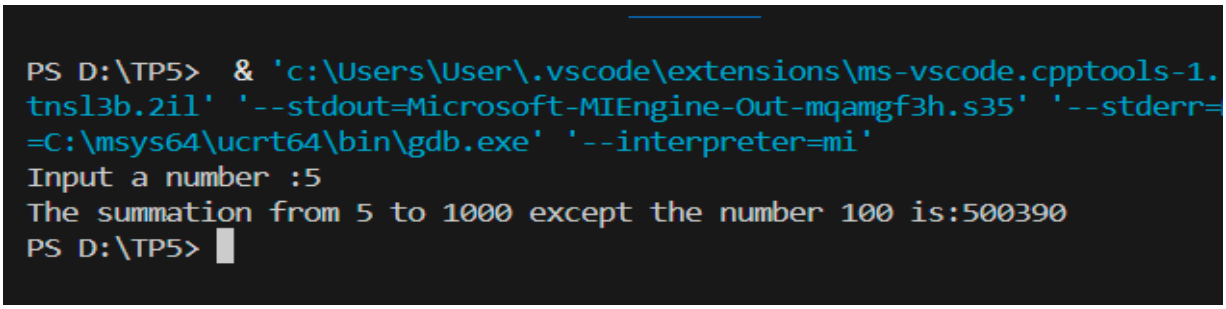
- **Loop**

  - `for(int i=n; i<=1000; i++)` → goes from `n` up to `1000`.
  - Inside loop:
  - If `i == 100`, the `continue` statement skips that iteration (so 100 is not added).
  - Otherwise, `sum = sum + i` adds the number to the total.

- **Output**

- After loop finishes, program prints the result.

Result:



Problem3: Write a program to display all even numbers between 1 and 50 using a for loop.

Examples:

Output:

The summation of all even numbers between 1 and 50: 2 + 4 + 6 + … 48 = **600**

Code

problem1.c | problem2.c | problem3.c | problem6.c | ex.c | ex5.

problem3.c > main()

```c
#include <stdio.h>
int main() {
    int sum = 0;

    printf("The summation of all even numbers between 1 to 50 is: ");

    for (int i = 2; i <= 50; i += 2) {
        sum += i;
        if (i == 2) {
            printf("%d", i);
        } else {
            printf("+%d", i);
        }
    }

    printf("=%d\n", sum);

    return 0;
}
```

- **Initialization**
  - **int sum = 0;**
  - sum starts at 0 and will accumulate the total of all even numbers.
- **Loop**
  - **for (int i = 2; i <= 50; i += 2)**
  - Starts at 2 (the first even number).
  - Increments by 2 each time (i += 2), so it only visits even numbers: 2, 4, 6, …, 50.

- **Accumulation**
  - **sum += i;**

  - Adds the current even number to sum.

- **Printing numbers**

  - If i == 2, print just 2 (no + before the first number).
  - Otherwise, print +i (so the output looks like 2+4+6+...+50).

- **Final output**

- **printf("=%d\n", sum);**

  - After the loop, prints the total sum with an = sign.

Key Points

- Using i += 2 is cleaner than looping through all numbers and checking if(i % 2 == 0).
- The if (i == 2) condition ensures the first number doesn't have a + sign before it.
- The final sum is **650**, which matches the formula for the sum of even numbers.

Result

```
PS D:\TP5>  & 'c:\Users\User\.vscode\extensions\ms-vscode.cpptools-1.28.3-win32-x64\debugAdapters\bin\WindowsDebugLauncher.exe' '--stdin=Microsoft-MIEngine-I
xeyvg5.zfx' '--stdout=Microsoft-MIEngine-Out-gzezyzbh.1as' '--stderr=Microsoft-MIEngine-Error-w0rxwooq.opg' '--pid=Microsoft-MIEngine-Pid-m1lxue2i.mk2' '--db
=C:\msys64\ucrt64\bin\gdb.exe' '--interpreter=mi'
The summation of all even numbers between 1 to 50 is: 2+4+6+8+10+12+14+16+18+20+22+24+26+28+30+32+34+36+38+40+42+44+46+48+50=650
PS D:\TP5>
```

Problem4: Create a program that calculates and prints the power of 2 for the first 10 natural numbers using a for loop.

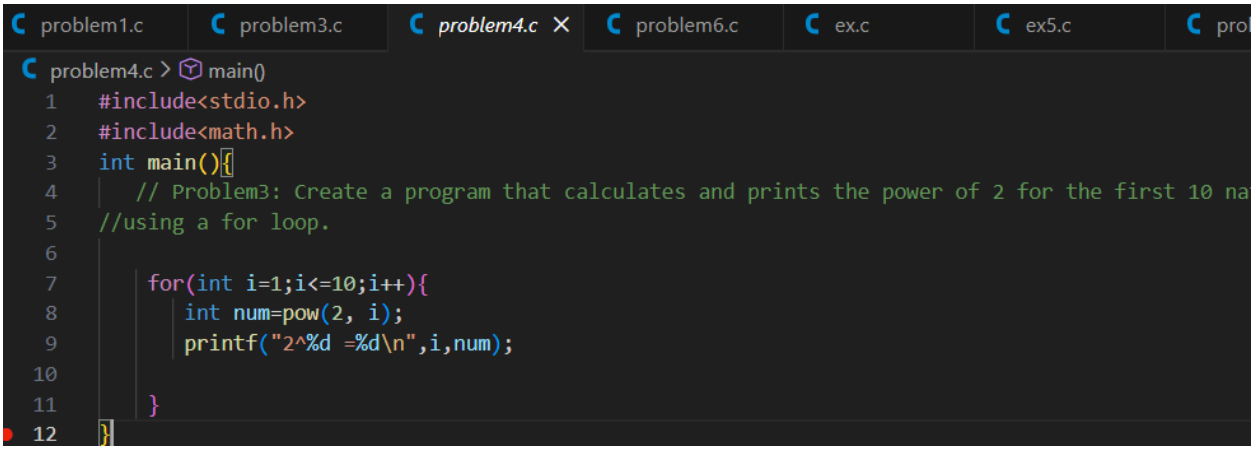
Examples:

Output:

The power of 2 for the first 10 natural numbers are:

$2^1 = 2$
$2^2 = 4$
$2^3 = 8$
$2^4 = 16$
$2^5 = 32$
$2^6 = 64$
$2^7 = 128$
$2^8 = 256$
$2^9 = 512$
$2^{10} = 1024$

Code



- **#include <stdio.h>**

This includes the **Standard Input Output** library so you can use printf() to display text.

- **#include <math.h>**

This includes the **Math library**, which gives you access to functions like pow() for exponentiation.

- int main() { ... }

This is the **main function** — the entry point of your program.

- **for (int i = 1; i <= 10; i++)**

This is a **for loop** that runs 10 times:

- Starts with i = 1
- Ends when i > 10
- Increments i by 1 each time

Each i represents the exponent in 2^i.

- **int num = pow(2, i);**

This calculates (2^i) using the pow() function:

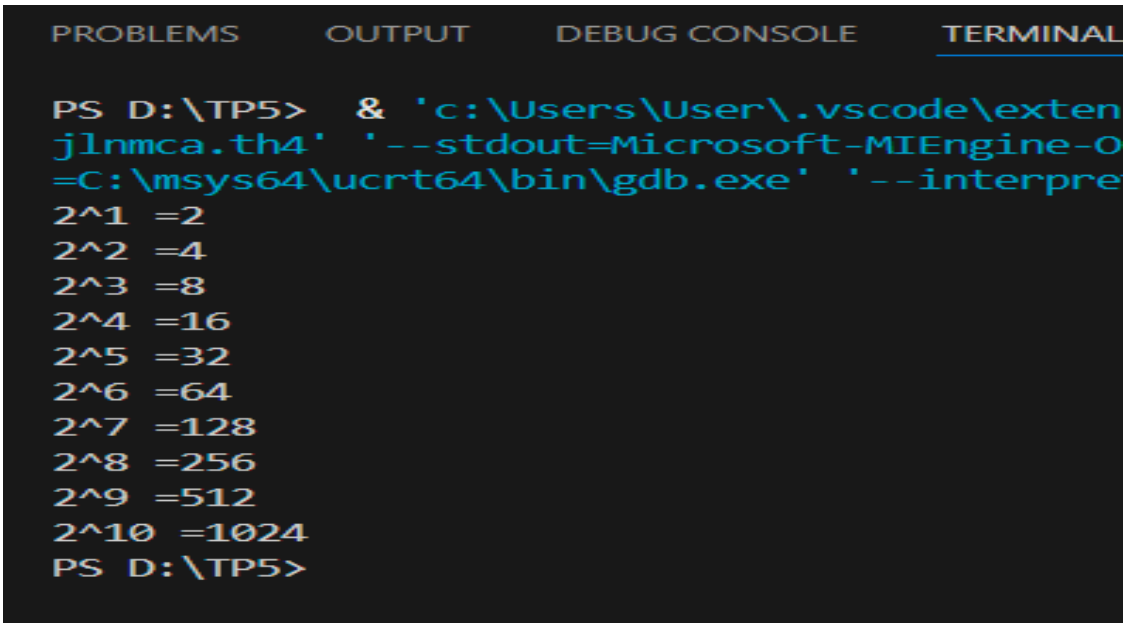
- pow(2, i) returns a floating-point number (a double)
- You store it in an int, which works fine here because powers of 2 are whole numbers

- **printf("2^%d = %d\n", i, num);**

This prints the result:

- %d is a placeholder for integers
- \n moves to the next line

Result:



Problem5: Create a program to ask a user to input a number. The program will tell the user whether it is a prime number or not.

INPUT:
Enter a number: 5
Output:
5 is a prime number because it is only divisible by itself and 1.
.....
INPUT:
Enter a number: 12
Output:
12 is not a prime number because it is not only divisible by itself and 1.

- `#include <stdio.h>`

- Includes the standard input/output library so you can use printf() and scanf().

- **`int num, count = 0;`**

- num → stores the number entered by the user.
- count → keeps track of how many divisors num has. Starts at 0.

- **`printf("Entert number :");`**

- Displays a message asking the user to enter a number.
  (You might want to fix the typo: "Enter number :")

- **`scanf("%d", &num);`**

- Reads the number entered by the user and stores it in num.

- **`for (int i = 1; i <= num; i++)`**

- Loops from 1 up to num.
  Each i is tested to see if it divides num.

- **`if (num % i == 0)`**

- Checks if i divides num exactly (no remainder).
  If true, then i is a divisor of num.

- **`count++;`**

- Increases the divisor count each time a divisor is found.

- **`if (count == 2)`**

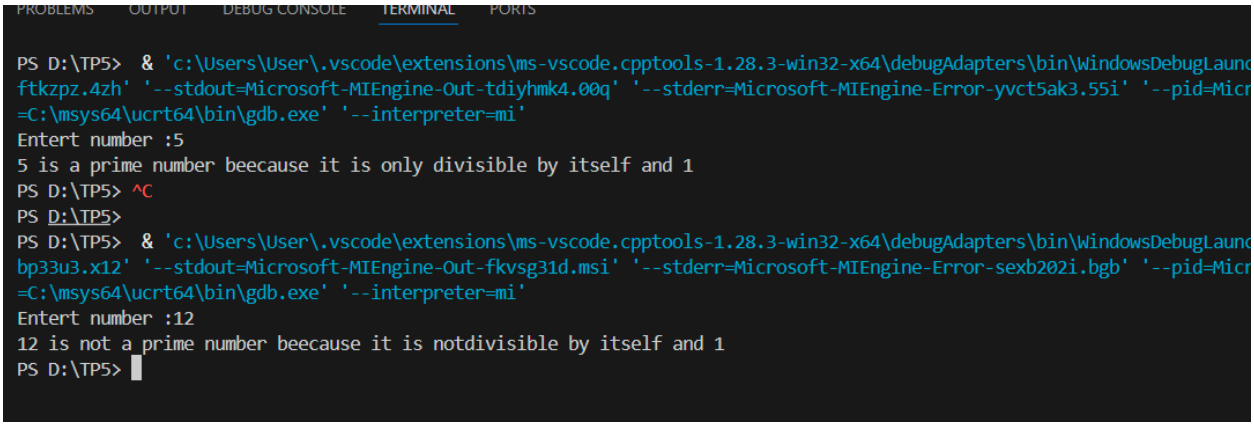
- After the loop finishes:

- If count == 2, then num has only two divisors (1 and itself) → **prime number**.
- Otherwise, it has more divisors → **not prime**.

Output

- If prime:
- 7 is a prime number because it is only divisible by itself and 1
- If not prime:
- 8 is not a prime number because it is divisible by more than just 1 and itself

Result:



Problem6: Create a program to identify and print all prime numbers between n and m using a for loop, where n and m are integer numbers input by the user.

INPUT:

Enter n: 20

Enter m: 50

Output:

All prime numbers between 20 and 50 are: **23,29,31,37,41,43 , 47.**

……

INPUT:

Enter n: 2

Enter m: 100

Output:

All prime numbers between 2 and 100 are: **2, 3, 5, 7, 11, 13, 17, 19, 23, 29, 31, 37, 41, 43, 47, 53, 59, 61, 67, 71, 73, 79, 83, 89, 97.**

Code

problem1.c | problem3.c | problem4.c | problem6.c | ex.c | ex5.c

problem6.c > main()

```c
#include <stdio.h>

int main() {
    int n, m, i, j, isPrime;
    printf("Enter n: ");
    scanf("%d", &n);
    printf("Enter m: ");
    scanf("%d", &m);
    printf("All prime numbers between %d and %d are: ", n, m);
    for (i = n; i <= m; i++) {
        if (i < 2)
            continue;
        isPrime = 1;
        for (j = 2; j < i; j++) {
            if (i % j == 0) {
                isPrime = 0;
                break;
            }
        }
        if (isPrime)
            printf("%d, ", i);
    }

    return 0;
}
```

- ## Input

The program asks the user to enter two numbers:

- `n`: the starting number
- `m`: the ending number

```
scanf("%d", &n);
scanf("%d", &m);
```

- ## Loop Through Each Number

It checks every number from `n` to `m` using a loop:

```
for (i = n; i <= m; i++)
```

○ Check if the Number is Prime

For each number `i`, it checks if it is divisible by any number from `2` to `i/2`:

```
for (j = 2; j <= i / 2; j++)
```

- If `i` is divisible by `j`, then it is **not prime**.
- If no such `j` is found, then `i` is **prime**.

This is done using a flag variable `isPrime`:

- Set to `1` (true) at the start.
- If a divisor is found, set to `0` (false).

○ Print the Prime Numbers

If `isPrime` is still `1` after checking, the number is printed:

```
if (isPrime)
    printf("%d, ", i);
```

○ Summary

- Uses two loops: one for each number, one for checking divisibility.
- Uses `isPrime` to track whether a number is prime.
- Prints all prime numbers between `n` and `m`.

Result:

