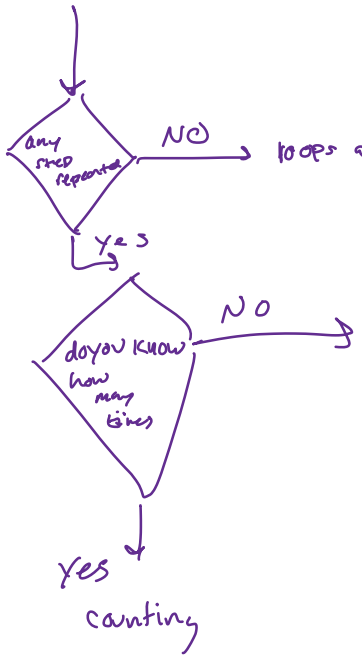
Loops

Want repeat a procedure over and over

↳ Can always do it Recursion

↳ recursion

↳ Φ turing machine doesn't have loops but you can still do everything



sentinel-controlled loop
endfile-controlled
General conditional loop
input validation loop

Counting Loop

While loops and for loops

while (condition) { "while ___ is true do { }, otherwise stop or never do it

3

```
int numberOfTimes = 0;
while (numberOfTimes < 10) {
    printf("hiii");
    numberOfTimes ++;
}
```

Make sure the variable in the conditions is being modified

3 → syntactic sugar

myVar++ → increment → myVar = myVar + 1;

myVar-- → decrement → myVar = myVar - 1;

myVar += 2 → myVar = myVar + 2

*=

/=

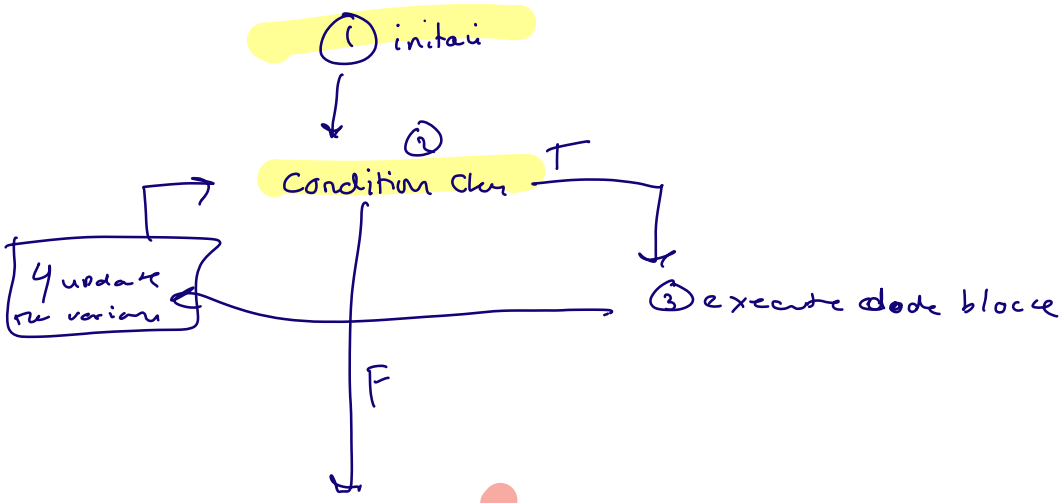
-=

```
for (int numberOfTimes=0 ; numberOfTimes < 10 ; numberOfTimes++ ) {
    printf("hi\n");
}
```

① variable init ② condition ③ ④ variable update

numberOfTimes: 0

3



ex) ans = $\sum_{i=1}^{n} i$

while

```
int computeSum(int n) {
    int counter = 1          (1)
    int sum = 0;
    while(counter <= n) {    (2)
        sum += counter;
        counter++;           (3)
    }
    return sum;
}
```

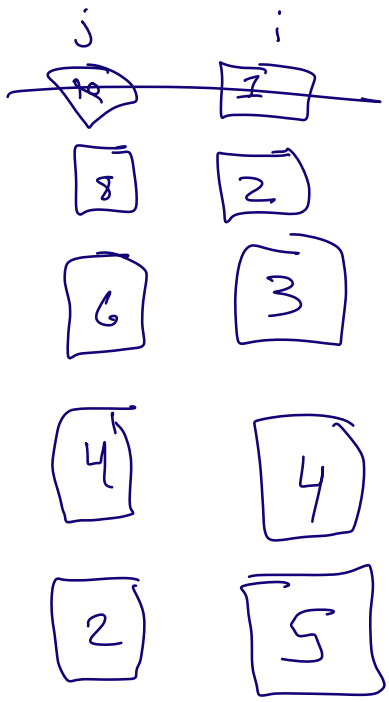
For

```
int ForcomputeSum(int n) {
    int sum = 0;
    for(int i = 1; i <= n; i++) {
        sum += i;
    }
    return sum;
}
```

3 if your for loop don't need brackets

example What does this do?

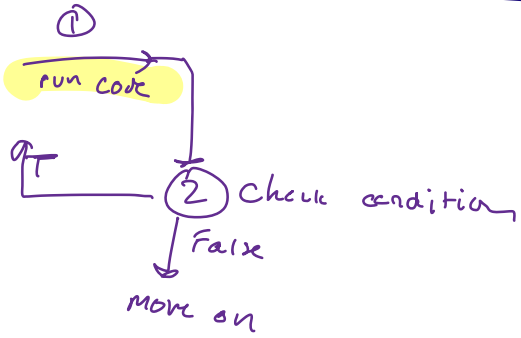
```
int j = 10;
for (int i = 1; i <= 5; i++) {
    printf("%d %d \n", i, j);
    j -= 2
}
```



OUTPUT

i	10
2	8
3	6
4	4
5	2

Conditional loop



//user input validation

ex

```
int numSamples;
do {
    printf("hey give us the number of samples:");
    scanf("%d", &numSamples);
} while (numSamples < 0);
```

do-while

---

bad: using while loop

```
printf("hey give us the number of samples:");
scanf("%d", &numSamples);
while (numSamples < 0) {
    printf("hey needs to be positive try again:");
    scanf("%d", &numSamples);
}
```

Debugging

Constants → is called a compiler directive

```
#define DEBUG 1

#define PI 3.14159
#define WINNING_SCORE 18

while ( score != WINNG_SCORE ) {
    sum += score
    if (DEBUG) {
        printf ("....%d", score );
    }
    //do other stuff
}
```

debugging block

Off-by-1

sum = 0 → $\sum_{i=0}^{n-1}$

```
for (int i = 0; i < n ; i++ ) {
    sum += i;
}
```

Example End File loop → accumulate numbers from

```
n = 0;
sum = 0;
status = scanf ("%d", &n)
while ( status == 1 ) {
    sum += n;
    status = scanf("%d", &n );
}
```

infinite loop without last line

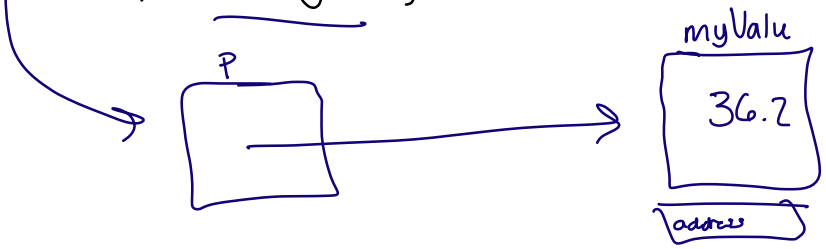
Pointer

&myValue ← gets the address of myValue

↳ give a type to an address.

We write pointers as

```
float myValue;
float *p; // this is a pointer. → float*
          // p is the address to a float.
p = &myValue;
```



What is the address p?

* ← dereferences a pointer to access that memory box

ex)

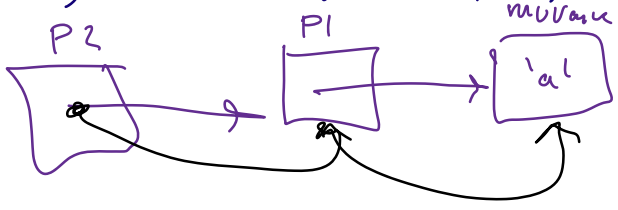
```
printf("the value of myValue is %of", *p);
```

↳ says follow the pointer and return value

```
//say you want update myVale by adding 1
*p = 1 + (*p)
```

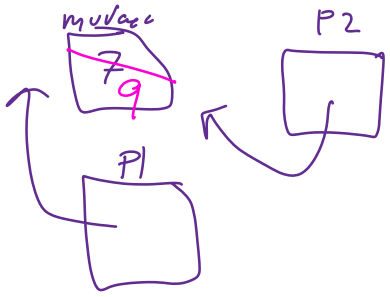
---

how can the value of the following variable myValue using only p2?




```
char myValue = 'a';
Char *p1 = &myValue;
Char **p2 = &p1;
printf("myValue is %c", **p2 );
```

```
float myVale = 7;
float *p1 = &myValue;
float *p2 = &myValue
*p1 = *p1 + 2;
printf("my vale is %of", *p2);
```

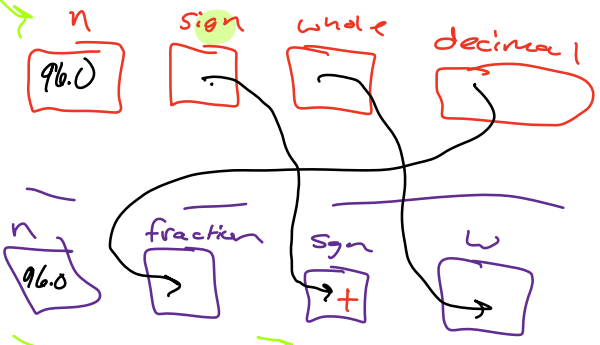


9

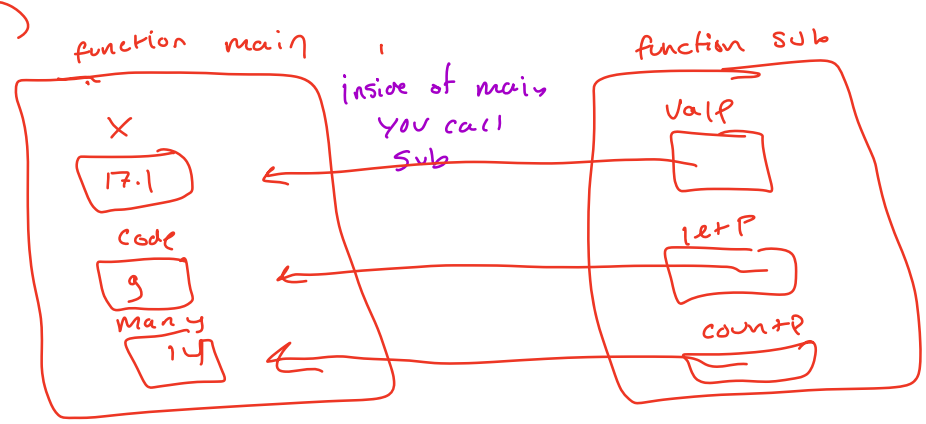
Multiple Returns

```
// is to seperate a double into the sign, whole number, decimal
void Seperate (double n, char *sign, int * whole, double *decimal) {
    if (n < 0) {
        *sign = '-';
    } else if (n == 0) {
        *sign = ' ';
    } else {
        *sign = '+';
    }
    *whole = int(n);
    *decimal = n - (*whole);
}

int main () {
    double n = 96.0;
    int w;
    char sgn;
    double fraction;
    Seperate(n, &sgn, &w, &fraction)
    printf("%c %d.%of", sgn, w, fraction);
```



ex)



code	when legal	data type	value
valp	sub	double *	address of x
&many	main	int *	adress of many
code	main	char	'g'
&code	main	char *	adress of code
countp	sub	int *	adress of many
* countp	sub	int	many = 14
& countp	sub	int **	adress of adress countp.
* valp	sub	double	17.1
letp	sub	char *	address of code
&x	main	double *	addres of x

confusing:

//declare a pointer

char *p; → type: char pointer

//outside a declaration as function signature or return type

*p → dereference to pointer so *p has type char

&p → gets the address which is a pointer char **

has type