Updating Operators

Binary Opperators

*=
+=
-=
/=
%=

lowest level of presidence

Ex.

x+=4; this is same as x= x+4

Unary Opperators

++	plus plus
--	minus minus
X++	postfix
++X	prefix

Incriment/deremont by one value

Ex.

x=5

Z= x--

decrements by one so x=4

* when its posfix its sets value first then changes it. When it prefix it changes it then sets the value

Controll Structures

Selection Structures

```
if (condition=true)
  {
     do something
  }
else
  {
     Statement...
  }
```

The if Statement

The else Statement

Multiple Selection

```
If (condition) {
  // code
}
else if (condition){
  //code
}
else if (condition){
  // code
}
else (condition){
  // code

}
```

Nested Block

```
if(condition){

    if (condition){
        // code
    }

}
```

Repitition Structures

Loops repeat a block of statements for a determined amount of time.

- While loop
- do... while loop
- for loop

While loop

- pre condition

```
while (test condition){
    // code
}
```

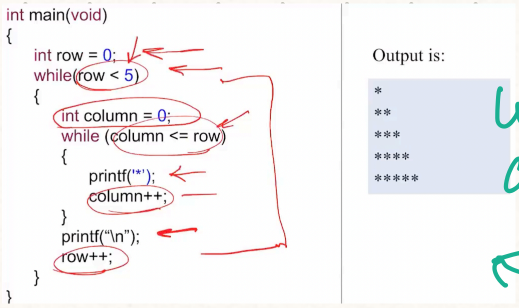
ex.

```
int i=0
while(i<5)
{
    printf("%d/n", i);
    i++;
}
```

Loop repeats till value of i = 5

0
1
2
3
4

Nested While Loops are permitted



```
int main(void)
{
    int row = 0;
    while(row < 5)
    {
        int column = 0;
        while (column <= row)
        {
            printf('*');
            column++;
        }
        printf("\n");
        row++;
    }
}
```

Output is:

```
*
**
***
****
*****
```

DO... While... Loop

```
do
{
    //Code...
} while (test_condition).
```

Ex.

```
do
{
    printf("%d/n",i);
    i++
}while(i<5);
```

Output: 0
1
2
3
4

Similar to while loop but backward. good for nested implimentation.

For Loop

- A pre tested loop

```
for( inicialization_expression; Test_Expression; Updating_expression)
  {
   //code
  }
```

-Stops if test_condition turns false

Ex...

```
for(i=0; i<5; i++)
  {
   printf("%d/n", i);
  }
```

Output: 0
1
2
3
4

You can initialize & declare a variable in a for loop but its domain is only in the for loop.

When initializing, Seperate with a comma   int i=0, j=5;

Jump Structures

- break; → Stops a loop & jumps to code after loop
- Continue; → will take you back to begining of a loop
- goto (not recommended)

Functions

Pre-defined functions

- ex. math Library

$|x|$ → fabs(x)

$|2*8|$ → fabs(x)

x^3 → pow(x, 3)

e^{x+2} → exp(x+2)

include<math.h>

-Functions can return a maximum of one value

User defined functions

Three Concepts

- Function prototype or function delaration
  - Name of function
- Function definition
  - Internal algorythm in a function
- Function call
  - Call a function in the main body

Function Prototype:

```
double average(int x, int y, int z); {
```

return value

name

arguments

Function name & local variable name can be same

Casting

Definition

```
    double average = (double)(x+y+z)/3.0;
    return average;
}
```

★ prototype must be before main but function can be after main

Prototype Argument names do not have to match definition argument names

Memory Allocation Space

- Global Declaration → defined outside of a function
- Local / Automatic Declaration ↳ must be initialized or it will be 0

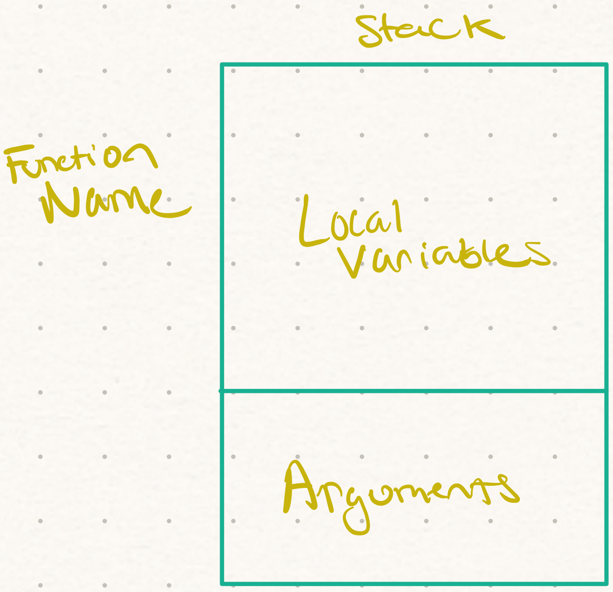
Argument of A Function

↑ must be initialized or it's value will be garbage

Activation Record (AR)

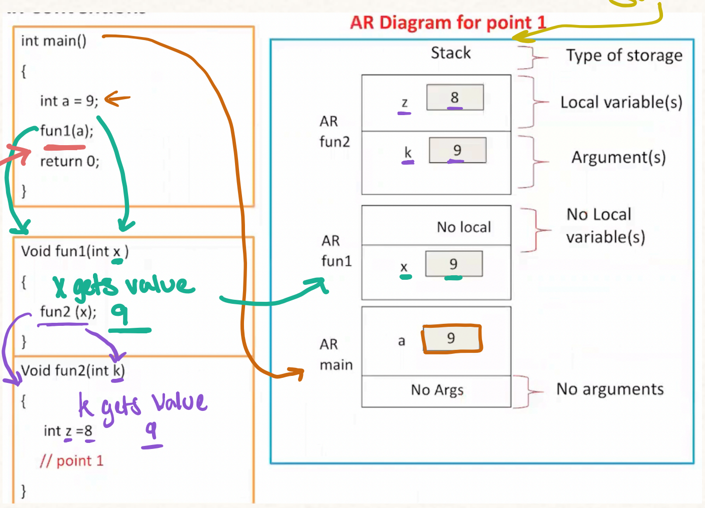
Graphical Representation of variables of an active function on the memory

An activation record keeps track of which functions are currently active & not terminated



Activation records should be drawn bottom to top in the order that they run

Functions only allocated on stack



```
int main()
{
    int a = 9;
    fun1(a);
    return 0;
}

Void fun1(int x )
{
    fun2 (x);
}

Void fun2(int k)
{
    int z =8
    // point 1
}
```

If you draw diagram here you only show main function

Memory Address

- Memory address of a variable is same as address Of first byte.

  Ex. 1000 1001 1002 1003

  This is like floor building numbers

"Address of" Opperator

This is &

printf("The address of myChar is %p\n", &myChar);

Can us %p or %lu (unassigned long integer)

Pointer

. A pointer is a data type that can hold the address of another variable of the same type.

declare: type* variable_name;

Ex. *p1 = 50

↙

derefrencing Opperator

Ex. int* p1;

p1 = &x p2 = &x

(puting value of 50 in location p1 is pointing to)

int* c = Null;

points to nowhere

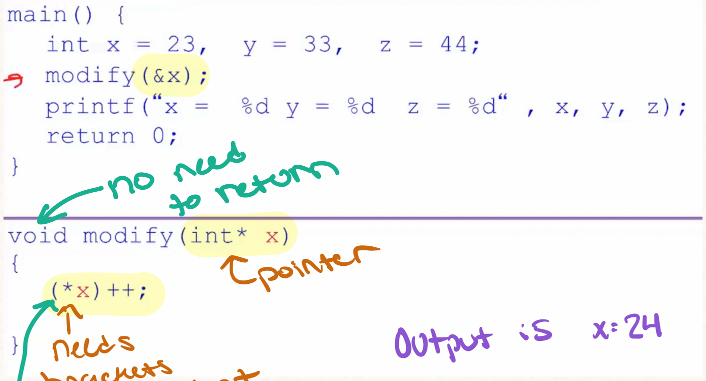
Pointers As Function Arg

Data Can be passed into functions by address

- When passed by address the function has access to the data (Function can modify variable)

Example

↓



```
main() {
   int x = 23,   y = 33,   z = 44;
   modify(&x);
   printf(“x =  %d y = %d  z = %d“ , x, y, z);
   return 0;
}
```

no need to return

```
void modify(int* x)
{
   (*x)++;

}
```

pointer

needs brackets low precident

must dereference

Output is x:24

Good for functions when need to return multiple values & cannot