Functions, vars, operators, loops, modularity

September 6, 2019 7:52 AM

Variable Swap

```
int temp = var1;
var1=var2;
var2=temp;
```

Operator precedence

```
int a = 16;
int b = 4;
double c = 1.5;
double d = c + a * b
```

- d = 65.5, double, since has a double in eqn

```
double d = b/a
```

- d = 0, int since casted to int, 0.25 -> 0

```
double d = (double) b/a
```

- cast b to double, d = 0.25

Boolean logic

- && need both true
- || need one of the two true

Indentation

```
int r;g
int a = -5;
int b = 6;
if (a<0 || b>0)
        r=1;
else
        r=2;
        a=0;
```

- a = 0, b = 6, r = 1

Loops

```
int i =1;
int j = 0;
while(i < 5 && j < 4) {
        j = j+1;
        i++;
}
printf("i = %d, j = %d", i, j);
```

- i = 4, j = 6
- **Nested Loops**
  - *coun1 = 3, count 2=9*

Functions

- params or Input stream data --> Funciton --> output stream data or return
  - **Params**
    - Actual param: spec by function caller
    - Formal param: found in signature of function itself

    - Params passed by value, value of param copied to formal param
    - actual and formal params different variables in memory
- Param scopes global vs local different

Arrays

- collection of data elements of same type
- stored in consecutive memory locations, each element referenced by and index
- declared liek ordinary var, with [] and size
- can be initialised when declared, or using a loop, from input, or other, cannot be assigned to existing array
- array can be passed in function like an array variable
    - size usually passed as additional var

Function parameters

- params passed by reference
    - address (rather than value) of actual parameter is copied to formal parameter when function is called
    - Making a change to value of formal parameter effectively changes value of actual parameter
    - same thing occurs with array params, passed by reference

Multi-dimensional arrays

- stores 2D dimensional array contiguously like 1D array
  void myfunction(int data[][NUMCOLS], int numrows);

Addresses and Pointers

- Every storage location in memory RAM has an address associated
    - address is memory location where var identifier stores data
    - like a mailbox number, access contents value

Variable declaration

- Each byte has unique address
- when compiling, compiler knows how much memory to allocate to each variable
    - ex. 4 bytes for int, 1 byte for char
- address with *scanf* requires provide address of locatin using address operator &
    - ex. *scanf("%d", &a)*
    - scanf modifies value fo var a defined outside scanf's call stack
- pointer is a data type that contains address of object in memory
    - ex. *int a = 5; int* p = &a;*
    - p is a pointer variable storing address of a