ESTR1002 Review

Note: this is just a review; you should look at the examples and details in lectures notes and textbook, etc. etc.

Lec02: Language Basics & Data Type

- Review: 02a
  - Operators: Binary, Unary & Ternary
  - Mind the Division (whatever language)!!!!
    - Integer division -> int in, int out, e.g., 1 / 2 = ?
    - Division by zero error: 1/0 and 1%0
  - Operator Precedence & Associativity: () is the King!
  - Shortforms: +=, -=, *=, /= and %=
  - ++ and – operators; prefix or postfix: i++ and ++i
  - swap data between two variables
- Review: 02b
  - int: exact, range, diff. types, overflow & underflow
  - double: may not be exact, but has more detail
  - Conversion: implicit and explicit conversions; i = (int) f;

Lec03: Control Flow (part 1)

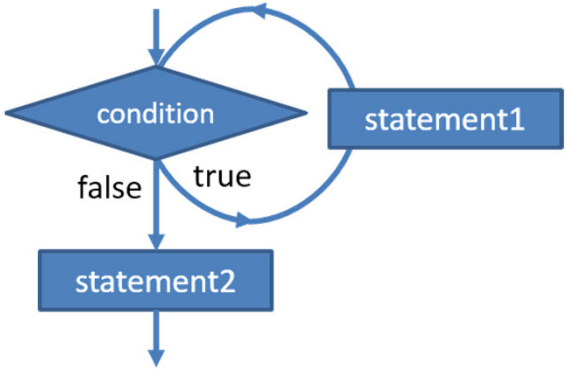
- Relational operators: ==, !=, >, >=, etc.
  - X == Y is an exact comparison; beware: floating pt.
- Logical operators: !, && and ||
  - Boolean expressions in C evaluate to 1 (T) / 0 (F)
  - A zero is treated as false
  - Any non-zero value is treated as true
- IF statement:
  - Need { } for compound statement
  - If else -> mutually exclusive cases
- Indent your code please!!!
- Common mistakes: if (a = 0) and if (i = 0) ;

```
if ( expr )
     statement ;
next_statement ;
```

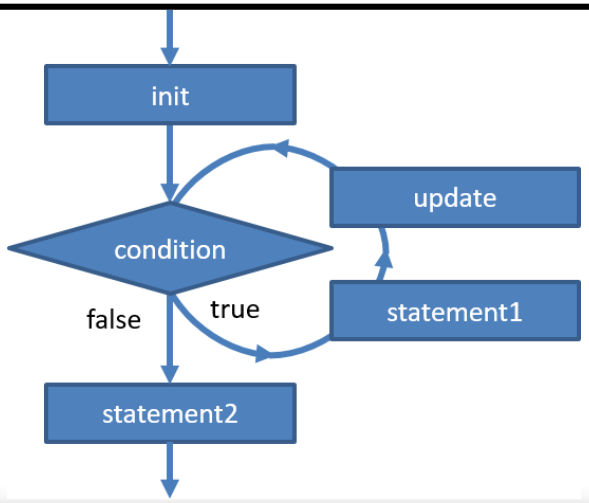
Lec04: Control Flow (part 2) - 1

- Looping:
  repeat loop body as
  long as condition is true
- **while loop**:
  - repeat zero or more times
  - good: sentinel-controlled loop
- **for loop**:
  - "init" always execute
  - "update" after loop body
  - good: counter-controlled loop
  - When some component missing?
- Additional: **Nested** loops

```
while ( condition )
    statement1 ;
statement2 ;
```



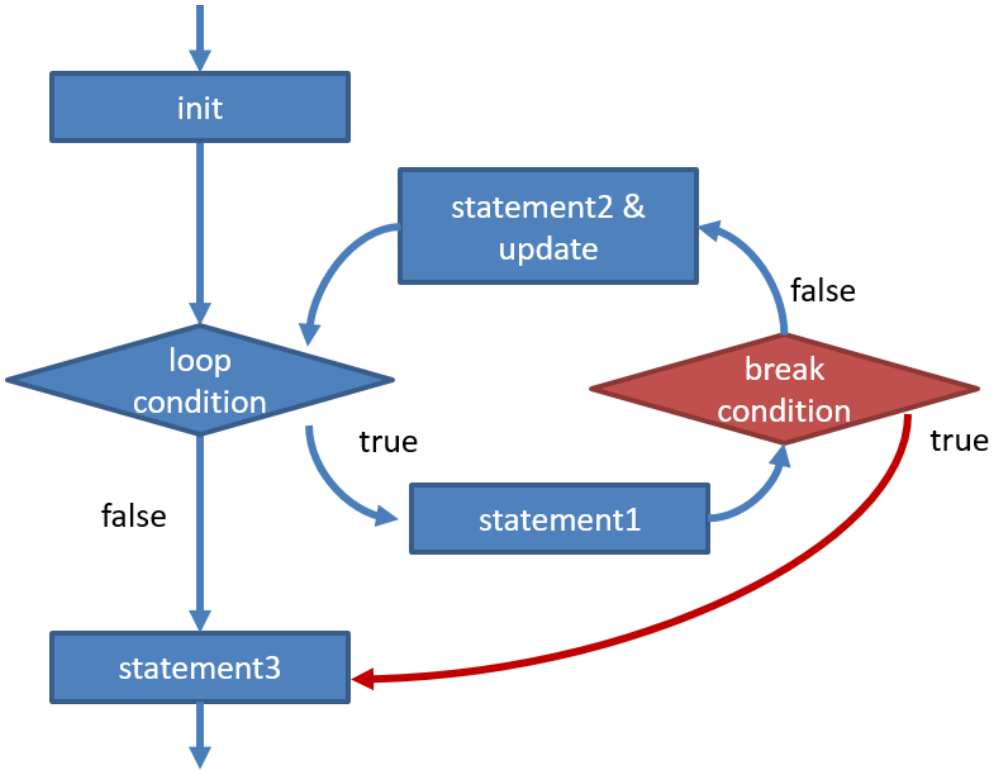

```
for ( init ; condition ; update )
    statement1 ;
statement2 ;
```



Lec04: Control Flow (part 2) - 2

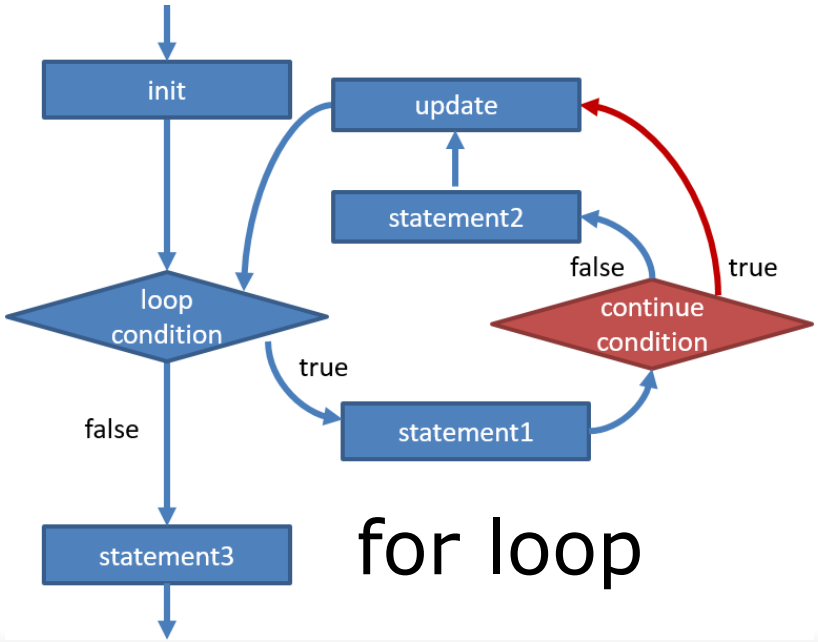
Interrupt the normal flow in a loop: break & continue

- break: when executed, causes the program to leave the **closest enclosing loop** immediately.

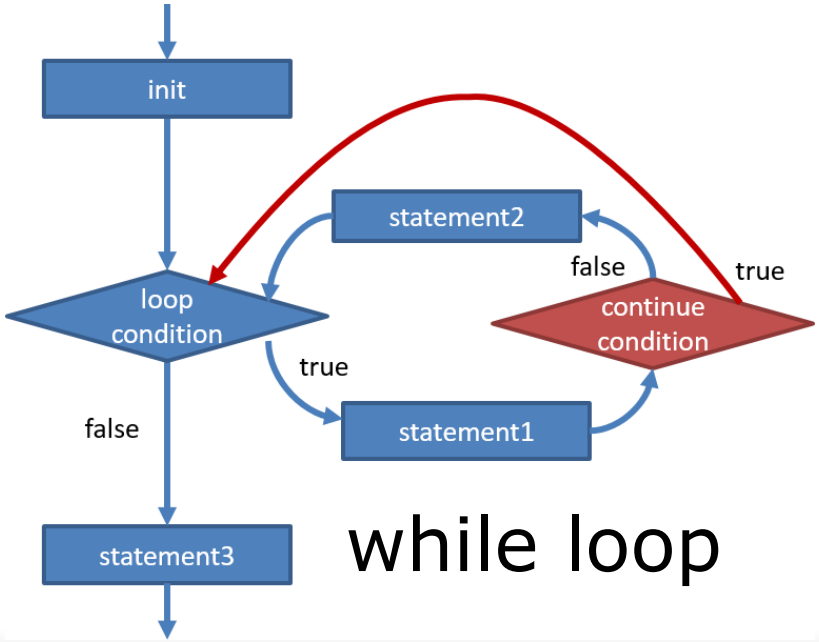


Lec04: Control Flow (part 2) - 3

- continue : when executed, causes the program to **skip the remaining statements** in the **closest enclosing loop** for the current iteration, and
  - **jump to** update (for loop)
  - **jump to** loop condition test (while loop)



for loop



while loop

Lec05: Array and Loops

- Definition of Array (in C):
  - An ordered list of data values: index [0, length-1]
  - Homogeneous (all elements - same data type)
- Syntax
  - Create an array and initialize the array contents
  - Access array element: read & modify
  - Multi-dimensional array: 2D, 3D, etc.
- Pay attention:
  - Runtime error: "array index out-of-bound" may happen
  - Array creation: constant (const) #elements (static alloc.)
  - Array copy – use a for loop
  - Array comparison – use a for loop
- Test for Prime & Count Prime Numbers; Sieve Algo.

Lec06: Character and String #1

- Character (char)
  - A char is also a number: int i = 'h' ; data[ch-'a']++;
  - ASCII table:
    - An ordered list of characters
    - A character encoding table with indices
  - Character I/O
    - getchar, scanf, putchar, printf
    - Comparison: <, <=, ==, >, >=
    - Ctrl-C, EOF, Ctrl-Z/Ctrl-D
  - Character Operations (need #include <ctype.h>):
    - isascii, isdigit, islower, isalpha, isspace, …
    - toupper, tolower

Lec06: Character and String #2

- What is a String?
  - An array (ordered list) of characters
  - Last character: '\0' is called the null terminator (allocate more memory for '\0')
- String operations (need #include <string.h>):
  - strlen – number of characters before '\0'
  - strcmp – return <0, ==0, or >0
  - strcpy – string copy
- String operation – I/O:
  - fgets - Perfect for reading input line by line

    (beware of the \n that is included in the output)
  - scanf - Perfect for reading input word by word
  - getchar – read char by char
- Note: the cheat sheet at the end is very useful!!!

Note: you may learn memcpy, memset, etc. yourself (byte by byte)

Lec07: Functions

- A sequence of statements for performing a **specific task**
- Advantages:
  - Readable, Reusable, divide-and-conquer, larger software
- **Function definition**: **<return type, name, params, body>**
  - Use meaningful function names
- **IN**: Pass data to function:
  - **Formal and Actual Parameters** & syntax for parameter list
  - Match by parameter positions
- **OUT**: Obtain data from function: "**return**"
  - All paths leaving the function must return a value UNLESS void
- **Function declaration** or **Function prototype**
  - like a contract: return type, function name, IN param type(s)
  - Why need it? Avoid ordering of func. & separate source files

Lec07: Algo. Part 1 – Searching

- What algorithms that you've learnt?
  - Finding min./max. in unsorted data (P.4 in notes)

    Idea: use a variable to remember the min./max seen so far
  - Selection sort algorithm (P.12 in notes)

    Idea: outer loop index "i" from 1st to (n-1)th location in data and inner loop keeps putting min. value in [i,n] at i-th location
  - Binary search for sorted data (P.26 in notes)

    Idea: check data@midpt & iteratively half the search space
- Other things that you've learnt:
  - Ternary operator: `min = ( a < b ) ? a : b ;`
  - Pass an array to function (P.15 in notes)
  - Linear search (order n) vs binary search (order $\log_2 n$)
  - Best/worst/average case analysis

Lec08: Variable Scope

- The **scope** of a variable determines **where** the variable can be accessed or used in a program.
  - **Local scope** (or Block scope) -> local variables
  - **Global scope** (or File scope) -> global variables
  - **Masking** concept:

    when multiple variables of same name are accessible, which one is being used?
  - Avoid global variables, e.g., by parameter passing
- The **storage class** of a variable defines its **life-time in computer memory** during the program execution.
  - **auto** (default) -> within a scope, e.g., function
  - **static** -> stays in memory until program finishes

Lec08: Algo. Part 2 – Recursion

- By writing a function that may call itself, we can
  Solve a problem by solving smaller versions of same problem
  - Beware: termination condition; otherwise, infinite loop
  - Beware: stack overflow – local memory for each function call
- Technique: Speedup through **Memorization**
  - Key Idea: **avoid re-computation** of the same case
  - Initialize a global array to remember the computed values
- Iteration vs Recursion: implement recursion by while/for
- Design recursion function before you start coding
  - Identify and **Model a problem with sub-problems**
  - Design function prototype for **data passing and return**
  - Design the **Termination** condition
- Examples: Lattice Path, Number Pyramid & Tower of Hanoi

Lec08: Random Num. Generator

- **Bitwise operators in C**
  - &, |, ^, ~, <<, and >>
  - XOR property: E = Data ^ Key and Data = E ^ Key
- Basic concept:
  - **Pseudo random number generator**: rand()
    - Random number seq.: A list of predictable numbers
    - Return a value inside [0, RAND_MAX]
  - **RAND_MAX**: a constant defined in stdlib.h
  - **srand(seed)**: reposition start point of the sequence
  - **srand(time(null))**: use current time as seed value
- **Monte Carlo** Simulation vs. **Las Vegas** Simulation
- **Algo. to shuffle cards**: N! outcomes, with equal prob.

Lec09: Algo. Part 3 – Permutation

Three scenarios:

- **Permutation with repetitions**: ab, aa, ba, bb
  - Using recursion, do not need multiple nested "for loops"
- **Permutation without repetitions**: ab, ba
  - Use an array - memorize the selected alphabets to avoid reselecting the same item
  - Remember to de-select after the recursion goes back
- **Combinations (no repeat and ordered)**: ab
  - Use a variable to remember previously-selected item, the next selected item should be > previously-selected item

Speed-up technique: memorization

e.g., N-Queen, N-Rook, Sudoku, etc.

Lec09: Structures-1

- A collection of related storage elements under a single name for better data organization:

```
struct struct_name
{
  <data type> member_variable_name ;
  <data type> member_variable_name ;
  ...
} ;   // don't miss this
```

- The dot operator (.) - member selection operator

```
Date today ;
today.day = 11 ;
today.month = 11 ;
```

- Alias (another name): typedef

```
typedef struct date Date ; // 3 ways to do so
```

Lec09: Structures-2

- Syntax #1: Initialize a structure:

```
Date xmas = { 25 , 12 , 2022 };
```

- Syntax #2: Copy a structure variable:

```
Date today ;
today = xmas ;
```

- Syntax #3: Pass a structure to a func. (pass by value):

```
void printDate( Date * d ) { ... (*d).day ...}
```

- Syntax #4: Pass a structure to a func. (pass pointer):

```
void printDate( Date * d ) { ... d->day ... }
```

- Syntax #5: Return a structure from a function:

```
Date readDate() { ... }
```

Lec11: Pointers-1

- A variable that stores memory address
- Five basic syntax:
  - Address-of operator – &

    ```
    int a = 1002 ; printf( "%p\n" , & a );
    ```
  - Create pointer variables

    ```
    int * ptr = & a ;
    ```
  - Dereference operator – *

    ```
    *ptr = 3260 ; printf( "%d\n" , *ptr );
    ```
  - Pointer to different data types are incompatible!
  - Good habit: assign NULL to pointers that point to nothing

    ```
    int * ptr = NULL ;
    ```
- Passing pointers to a function **emulates** the effect of "pass by reference"

Lec11: Pointers-2

- Additional topics:
  - Common Pitfalls, Dangling Ptr, Memory Keak: see notes
  - Pointer to Pointer: `int **ptr2 = & ptr ;`
  - Pointer and Array:

    Address of an array == Address of 1st element (base address)
    Address of arr[i] = base address + i × sizeof(element type)
    Array is like a constant pointer (always point to same address)
  - 2D Array has two levels of 1D arrays

```
int board[4][4] ;
// board is an array of four pointers (int *)
// board[i] is an array of four values (int)
```

  - Dynamic memory allocation: `malloc` and `free`

```
int * arr2 = (int *) malloc( sizeof(int) * 100 );
free( arr2 );
```

  - Pointer Arithmetic, Pointer & Structure, and Constant Pointers

Lec12: File I/O

- Compare Keyboard input VS File input:
  - Similarity: data read in a FIFO (first-in first-out)
  - Keyboard (wait for user) vs File: data readily available
- Three stages in File I/O (Reading/Writing):
  - Open file: FILE * fp = fopen(filename , opening mode)
    - ‘r’, ‘w’, ‘a’, etc.
  - Read/Write data:
    - Read: fscanf, fgetc, fgets
    - Write: fprintf, fputc, fputs
    - Note: fgets contains trailing \n and EOF (end-of-file)
  - Close file: fclose(fp)
- Three streams in <stdio.h>: stdin, stdout and stderr
- Other useful func.: feof(), ftell(), fseek(), rewind(), tmpfile()

Lec12: Formatted output (printf)

- Integer: %[flags][width]d

  [flags]:

  - + Print the plus sign (+) for non-negative numbers.
  - - Left-justify; default is right-justification

  [width] :

  - Minimum number of characters to be printed

- Floating point number: %[flags][width][.precision]f

  [flags]: same as above

  [width]: same as above

  [.precision]: Number of digits to be printed after the decimal point (default is six)

  Note: please try the examples in the lecture notes