SFU

# Array Comparison, Strings and Loops

CMPT 125
Mo Chen
SFU Computing Science
17/1/2020

# Lecture 5

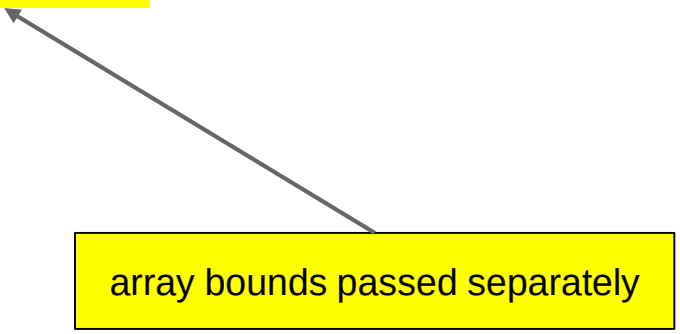
Today

- Array Comparison
- Strings
- Nested Loops

# Array Comparison

- Write a function to compare two arrays
- Array parameters passed by base address
  - Style points: use `int arr[]` instead of `int *arr`

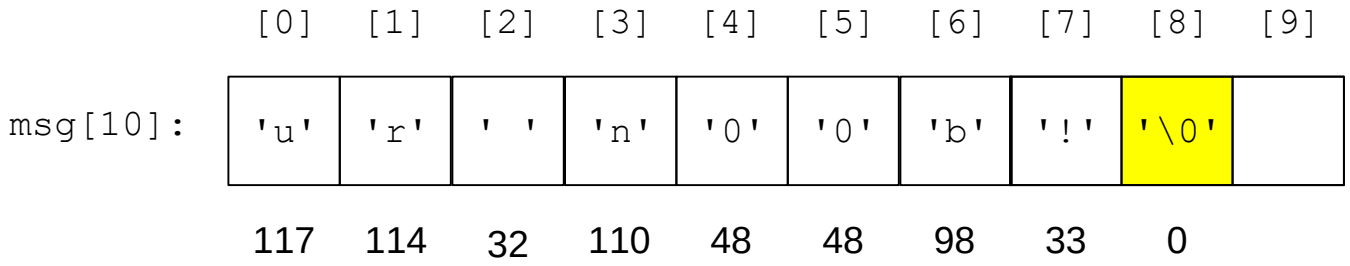
```
int arrCompare(int A[], int B[], int length) {
  for (int i = 0; i < length; i++) {
    if (A[i] < B[i]) {
      return -1;
    } else if (A[i] > B[i]) {
      return 1;
    }
  }
  return 0;
}
```



# Arrays of `char`

- type `char` is 1 byte per element
  - traditionally to hold one ASCII character
  - an array of `char` is a string!
- end of string terminated by *null char*: `'\0'`

```
int main ( ) {
  char msg[10] = "ur n00b!";
  printf("%s\n", msg);
}
```



| Dec | Hx | Oct | Char | Dec | Hx | Oct | Html | Chr | Dec | Hx | Oct | Html | Chr | Dec | Hx | Oct | Html | Chr |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 0 | 0 | 000 | NUL (null) | 32 | 20 | 040 |   | Space | 64 | 40 | 100 | @ | @ | 96 | 60 | 140 | ` | ` |
| 1 | 1 | 001 | SOH (start of heading) | 33 | 21 | 041 | ! | ! | 65 | 41 | 101 | A | A | 97 | 61 | 141 | a | a |
| 2 | 2 | 002 | STX (start of text) | 34 | 22 | 042 | " | " | 66 | 42 | 102 | B | B | 98 | 62 | 142 | b | b |
| 3 | 3 | 003 | ETX (end of text) | 35 | 23 | 043 | # | # | 67 | 43 | 103 | C | C | 99 | 63 | 143 | c | c |
| 4 | 4 | 004 | EOT (end of transmission) | 36 | 24 | 044 | $ | $ | 68 | 44 | 104 | D | D | 100 | 64 | 144 | d | d |
| 5 | 5 | 005 | ENQ (enquiry) | 37 | 25 | 045 | % | % | 69 | 45 | 105 | E | E | 101 | 65 | 145 | e | e |
| 6 | 6 | 006 | ACK (acknowledge) | 38 | 26 | 046 | & | & | 70 | 46 | 106 | F | F | 102 | 66 | 146 | f | f |
| 7 | 7 | 007 | BEL (bell) | 39 | 27 | 047 | ' | ' | 71 | 47 | 107 | G | G | 103 | 67 | 147 | g | g |
| 8 | 8 | 010 | BS (backspace) | 40 | 28 | 050 | ( | ( | 72 | 48 | 110 | H | H | 104 | 68 | 150 | h | h |
| 9 | 9 | 011 | TAB (horizontal tab) | 41 | 29 | 051 | ) | ) | 73 | 49 | 111 | I | I | 105 | 69 | 151 | i | i |
| 10 | A | 012 | LF (NL line feed, new line) | 42 | 2A | 052 | * | * | 74 | 4A | 112 | J | J | 106 | 6A | 152 | j | j |
| 11 | B | 013 | VT (vertical tab) | 43 | 2B | 053 | + | + | 75 | 4B | 113 | K | K | 107 | 6B | 153 | k | k |
| 12 | C | 014 | FF (NP form feed, new page) | 44 | 2C | 054 | , | , | 76 | 4C | 114 | L | L | 108 | 6C | 154 | l | l |
| 13 | D | 015 | CR (carriage return) | 45 | 2D | 055 | - | - | 77 | 4D | 115 | M | M | 109 | 6D | 155 | m | m |
| 14 | E | 016 | SO (shift out) | 46 | 2E | 056 | . | . | 78 | 4E | 116 | N | N | 110 | 6E | 156 | n | n |
| 15 | F | 017 | SI (shift in) | 47 | 2F | 057 | / | / | 79 | 4F | 117 | O | O | 111 | 6F | 157 | o | o |
| 16 | 10 | 020 | DLE (data link escape) | 48 | 30 | 060 | 0 | 0 | 80 | 50 | 120 | P | P | 112 | 70 | 160 | p | p |
| 17 | 11 | 021 | DC1 (device control 1) | 49 | 31 | 061 | 1 | 1 | 81 | 51 | 121 | Q | Q | 113 | 71 | 161 | q | q |
| 18 | 12 | 022 | DC2 (device control 2) | 50 | 32 | 062 | 2 | 2 | 82 | 52 | 122 | R | R | 114 | 72 | 162 | r | r |
| 19 | 13 | 023 | DC3 (device control 3) | 51 | 33 | 063 | 3 | 3 | 83 | 53 | 123 | S | S | 115 | 73 | 163 | s | s |
| 20 | 14 | 024 | DC4 (device control 4) | 52 | 34 | 064 | 4 | 4 | 84 | 54 | 124 | T | T | 116 | 74 | 164 | t | t |
| 21 | 15 | 025 | NAK (negative acknowledge) | 53 | 35 | 065 | 5 | 5 | 85 | 55 | 125 | U | U | 117 | 75 | 165 | u | u |
| 22 | 16 | 026 | SYN (synchronous idle) | 54 | 36 | 066 | 6 | 6 | 86 | 56 | 126 | V | V | 118 | 76 | 166 | v | v |
| 23 | 17 | 027 | ETB (end of trans. block) | 55 | 37 | 067 | 7 | 7 | 87 | 57 | 127 | W | W | 119 | 77 | 167 | w | w |
| 24 | 18 | 030 | CAN (cancel) | 56 | 38 | 070 | 8 | 8 | 88 | 58 | 130 | X | X | 120 | 78 | 170 | x | x |
| 25 | 19 | 031 | EM (end of medium) | 57 | 39 | 071 | 9 | 9 | 89 | 59 | 131 | Y | Y | 121 | 79 | 171 | y | y |
| 26 | 1A | 032 | SUB (substitute) | 58 | 3A | 072 | : | : | 90 | 5A | 132 | Z | Z | 122 | 7A | 172 | z | z |
| 27 | 1B | 033 | ESC (escape) | 59 | 3B | 073 | ; | ; | 91 | 5B | 133 | [ | [ | 123 | 7B | 173 | { | { |
| 28 | 1C | 034 | FS (file separator) | 60 | 3C | 074 | < | < | 92 | 5C | 134 | \ | \ | 124 | 7C | 174 | | | \| |
| 29 | 1D | 035 | GS (group separator) | 61 | 3D | 075 | = | = | 93 | 5D | 135 | ] | ] | 125 | 7D | 175 | } | } |
| 30 | 1E | 036 | RS (record separator) | 62 | 3E | 076 | > | > | 94 | 5E | 136 | ^ | ^ | 126 | 7E | 176 | ~ | ~ |
| 31 | 1F | 037 | US (unit separator) | 63 | 3F | 077 | ? | ? | 95 | 5F | 137 | _ | _ | 127 | 7F | 177 | &#127; | DEL |

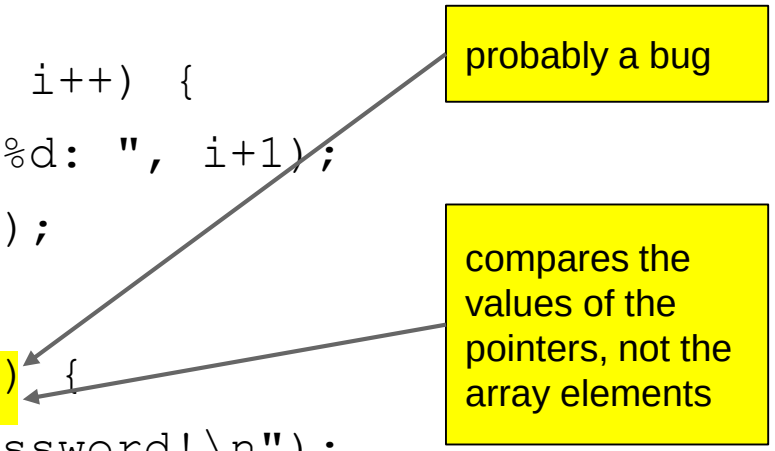
Source: www.LookupTables.com

# Array Comparison

Puzzle:  What's wrong with this code?

```
int main ( ) {
  int password[3] = {1,2,3};
  int answer[3];

  for (int i = 0; i < 3; i++) {
    printf("Enter digit %d: ", i+1);
    scanf("%d", answer+i);
  }
  if (password != answer) {
    printf("Incorrect password!\n");
  }
}
```



# String Comparison

```
#include <stdio.h>
#include <string.h>

int main ( ) {
   char password[4] = "abc";
   char answer[4];

   printf("Enter 3-character code: ");
   scanf("%s", answer);
   if (strcmp(password, answer) != 0) {
     printf("Incorrect password!\n");
   }
}
```

not `&answer` because `answer` is a pointer!

C library function to do string comparisons:

- 0 means equal
- < 0 means first < last
- > 0 means first > last

# Common String Functions

`int strlen(char s[])`

- returns the length of the string
- counts characters until null terminator
- Q: What happens if there is no null terminator?

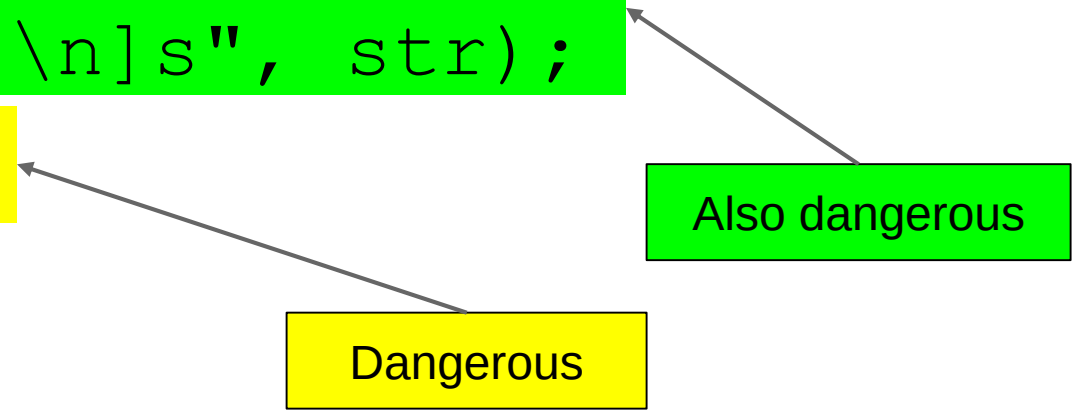
`void strcpy(char dest[], char src[])`

- copies the string `dest[]` ← `src[]`
- Q: What *must* be true about `dest[]`?

# String I/O

Input

- `scanf("%s", str);`
- `scanf("%[^\n]s", str);`
- `gets(str);`




Output

- `printf("%s", str);`
- `puts(str);`

# Nested Loops

- It is possible to include any sequence of statements within a loop body including:
  - calculations
  - function calls
  - if statements
  - other loops
- Just like you did in Python!

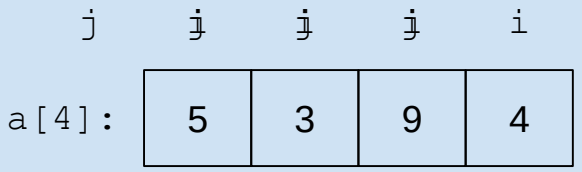
Classic Problem: Write a function that scans an array of `int`. It returns 1 if and only if two of the elements are the same, 0 otherwise.

# Classic Solution

```
int dup_chk(int a[], int length) {
  int i = length;
  while (i > 0) {
    i--;
    int j = i - 1;
    while (j >= 0) {
      if (a[i] == a[j]) {
        return 1;
      }
      j--;
    }
  }
  return 0;
}
```

Simulation:

dup_chk(a, 4):




These statements run the most frequently in the worst case

- What is the worst case?
- How many times when length = 4?

# Another Performance Measure

- Often consider the *worst-case* behaviour as a benchmark
  - make guarantees about code performance under all circumstances
- Can predict performance by counting the number of steps required by algorithm in the worst case
  - Derive total steps (T) as a function of input size (N)

# Analysis

```
int dup_chk(int a[], int length) {
   1  int i = length;
 N+1  while (i > 0) {
   N     i--;
   N     int j = i - 1;
 i+1     while (j >= 0) {
   i        if (a[i] == a[j]) {
              return 1;
            }
   i        j--;
          }
      }
   1  return 0;
}
```

Q. What is *N*?

- The number of elements in the array

Outside of loop: 2 (steps)

Outer loop: $3N + 1$

Inner loop: $3i + 1$ for all possible $i$ from 0 to $N - 1$.

$= 3/2\ N^2 - 1/2\ N$

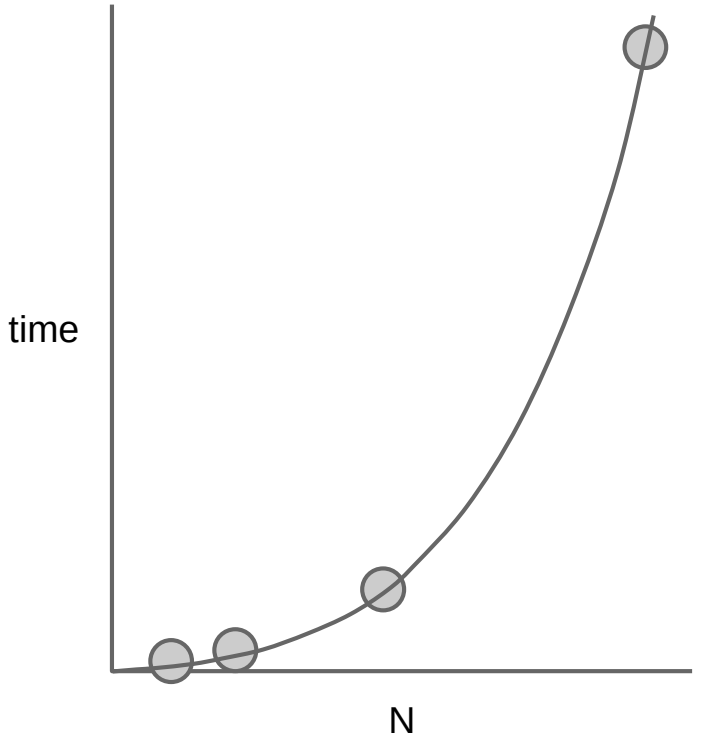
Grand total = $3/2\ N^2 + 5/2\ N + 3$

A *quadratic* function!

# Empirical Measurement

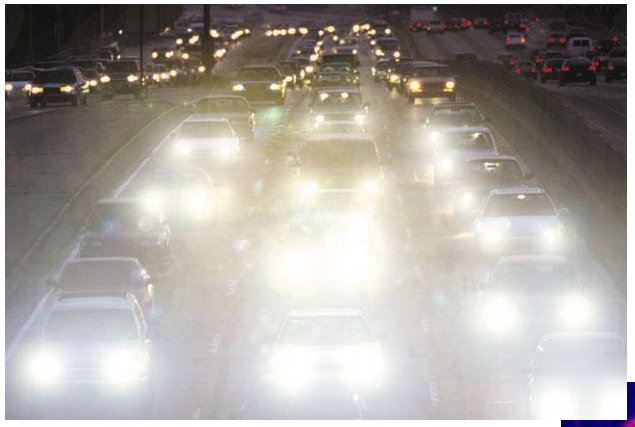
- Another graph - a quadratic this time!
- Confirms predictions: *doubling* (x2) the input size leads to **quadrupling** (x4) the running time

| N | time (in ms) |
|---|---|
| 10,000 | 89 |
| 20,000 | 365 |
| 40,000 | 1,424 |
| 100,000 | 9,011 |



# 2D Maximum Density Problem

Problem: Given a 2-dimensional array ($N \times N$) of integers, find the $10 \times 10$ swatch that yields the largest sum



Applications:

- Resource management and optimization
- Finding brightest areas of photos

# Algorithm / Code?

- Simple approach:  Try all possible positions for the upper left corner
  - $(N-9) \times (N-9)$ of them
  - use a nested loop
- add each swatch using a $10 \times 10$ nested loop
- A *brute-force* approach!
  - Generate a possible solution [naively]
  - Test it [naively]

# In C

Precise accounting:

$348N^2 - 6956N + 34762$ operations

```
int max10by10(int a[N][N]) {
  int best = 0;
  for (int u_row = 0; u_row < N-9; u_row++) {
    for (int u_col = 0; u_col < N-9; u_col++) {
      int total = 0;
      for (int row = u_row; row < u_row+10; row++) {
        for (int col = u_col; col < u_col+10; col++) {
          total += a[row][col];
        }
      }
      best = max(best, total);
    }
  }
  return best;
}
```

× (N − 9)

× (N − 9)

× 10

11

10

10

Approximate Method:

Count the *barometer instructions*, the instructions executed most frequently. Usually, in the innermost loop.

Innermost loop: $11 + 10 + 10 = 31$ ops

Total = $31 \times \mathbf{10} \times (\boldsymbol{N} - \mathbf{9}) \times (\boldsymbol{N} - \mathbf{9}) = \mathbf{310}\boldsymbol{N}^{\mathbf{2}}$