Computer Programming: Skills & Concepts (CP1)
Pattern matching with arrays; Bitwise operators

Last lecture
- Introduction to arrays.
- Using arrays for “character-statistics” on text.
- Relationship between arrays and pointers.
- Arrays as parameters to functions.

Today
- Strings.
- Arrays cont. - basic pattern matching.
- Bitwise operations on int (on board).

Basic data types in C
int char float double

Really that’s all . . .
except for variations such as signed char, unsigned char, short, . . .
What about strings?

In computer programming (all languages), a string is any sequence of characters.

- Many languages offer a string data type.
- C does not offer a string data type.
- A string is an array of char:
  - By C convention, strings end with a null character (0 or '\0').
  - Eg char month1[] = {'j','a','n','u','a','r','y','\0'};
  - Or (shorthand) char month1[] = "january";
  - In a function declaration, as in
    ```c
    int StringFoo(char line[], int length)
    ```
- Recall arrays as pointers; a string is also a pointer to char.


Pattern matching

We want to write a program that

- ask the user for a pattern
- filters subsequent input for that pattern

Template for reading input

```c
int c = getchar();
if (c == EOF) {
  return TRUE;
}
while (c != EOF) {
  /* do something */
  c = getchar();
}
```

Reading input line by line

We want to have a handy function GetLine that reads one line from input.

- How do we store the line of text?
- What is the stopping condition of the while loop?
- What happens inside the body?
**GetLine()**

Bool_t GetLine(char line[], int length) {
    int i = 0, c = getchar();
    if (c == EOF) return TRUE;
    while (c != '\n' && c != EOF) {
        if (i < length - 1) {
            line[i] = c;
            ++i;
        }
        c = getchar();
    }
    line[i] = '\0';
    return FALSE;
}

*NOTE* that GetLine assumes the array exists up to the given length – it does not create it.

**Substring matching**

- We know how to match characters
- How do we match a substring?

LINE: a test !
PAT: test

```
test
--> test <-- MATCH!
test
```

**The big picture**

```
char line[LINELength], pat[PAT_LENGTH];
GetLine(pat, PAT_LENGTH);
while (!GetLine(line, LINE_LENGTH)) {
    if (IsSubstringOf(pat, line)) {
        PutLine(line);
    }
}
```

**Matching condition**

First attempt:
```
int j = 0;
while (pat[j] != '\0' && pat[j] == text[start+j]) {
    ++j;
}
```

What happens if we run out of characters in text?

```
if (pat[j] == '\0') {
    return TRUE;
}
```
Matching condition

Improved:

```c
int j = 0;
while (pat[j] != '\0'
     && text[start+j] != '\0'
     && pat[j] == text[start+j]) {
    ++j;
}
if (pat[j] == '\0') {
    return TRUE;
}
```

Do we really need this?

Matching loop

```c
Bool_t IsSubstringOf(char pat[], char text[])
/* Returns TRUE iff pat is a substring of text. */
{
    int start = 0, j;
    while (text[start] != '\0') {
        /* match pattern starting at start */
        ++start;
        if (pat[j] == '\0') {
            return TRUE;
        }
    }
    return FALSE;
}
```