Today’s Goals

- **char**
- Input (getchar, scanf)
- Expressions
- Conditionals
  - if
  - switch
- Loops
  - while

### Section 1: 
**sizeof and Type Conversions**

- `sizeof(type)`
  - The `sizeof` operator returns the number of bytes required to store the given type
- Implicit conversions
- Explicit conversions
  - arithmetic
  - assignment
  - function parameters
  - function return type
  - promotion if possible

### Section 2: 
**Use of char (character)**

- Basic operations
  - Declaration: `char c;`
  - Assignment: `c = 'a';`
  - Reference: `c = c + 1;`

- Constants
  - Single-quoted character (only one)
  - Special characters: `\n`, `\t` (tab), `"` (double quote), `'` (single quote), `\` (backslash)

### Characters are Integers

- A `char` type represents an integer value from 0 to 255 (1 byte) or –128 to 127.
- A single quoted character is called a “character constant”.
- C characters use ASCII representation:
  - A' = 65 ... 'Z' = 'A' + 25 = 90
  - 'a' = 97 ... 'z' = 'a' + 25 = 122
  - '0' != 0 (48), '9' - '0' = 9
- Never make assumptions of `char` values
  - Always write 'A' instead of 65

### ASCII Table

American Standard Code for Information Interchange
A standard way of representing the alphabet, numbers, and symbols (in computers)

[wikipedia on ASCII](https://en.wikipedia.org/wiki/ASCII)

### char Input/Output

- Input
  - `char getchar()` receives/returns a character
  - Built-in function

- Output
  - `printf` with `%c` specification

```c
int main()
{
  char c;
  c = getchar();
  printf("Character >%c< has the value %d.\n", c, c);
  return 0;
}
```
The `scanf` Function

- Format string containing special symbols
  - `%d` for `int`
  - `%f` for `float`
  - `%lf` for `double`
  - `%c` for `char`
  - `
` for a newline
- List of variables (or expressions)
  - In the order corresponding to the `%` sequence

The function `scanf` is the input analog of `printf`

- Each variable in the list MUST be prefixed with an `&`.
- Ignores white spaces unless format string contains `%c`

Example:
```c
int main() {
  int x;
  printf("Enter a value:\n");
  scanf("%d", &x);
  printf("The value is %d.\n", x);
  return 0;
}
```

- White space in the format string match any amount of white space, including none, in the input.
- Leftover input characters, if any, including one `\n` remain in the input buffer, may be passed onto the next input function.
  - Use `getchar()` to consume extra characters
  - If the next input function is also `scanf`, it will ignore `\n` (and any white spaces).

Example with multiple variables:
```c
int main() {
  int x;
  char c;
  printf("Enter an int and a char: ");
  scanf("%d %c", &x, &c);
  printf("The values are %d, %c.\n", x, c);
  return 0;
}
```
**scanf Notes**

- Beware of combining `scanf` and `getchar()`.
- Use of multiple specifications can be both convenient and tricky.
  - Experiment!
- Remember to use the return value for error checking.

**if-else Statement**

```c
int main() {
    int choice;
    scanf("%d", &choice); //user input
    if (choice == 1) {
        printf("The choice was 1.\n");
    } else {
        printf("The choice wasn't 1.\n");
    }
    return 0;
}
```

**Expressions**

- Numeric constants and variables
  - E.g., 1, 1.23, x
- Value-returning functions
  - E.g., `getchar()`
- Expressions connected by an *operator*
  - E.g., `1 + 2`, `x * 2`, `getchar() - 1`
- All expressions have a type

**Boolean Expressions**

- C does not have type boolean
- False is represented by integer 0
- Any expression evaluates to non-zero is considered true
- True is typically represented by 1 however

**Conditional Expressions**

- **Equality/Inequality**
  - `if (x == 1)`
  - `if (x != 1)`
- **Relation**
  - `if (x > 0)`
  - `if (x >= 0)`
  - `if (x < 0)`
  - `if (x <= 0)`

**Assignment as Expression**

- **Assignment**
  - Assignments are expressions
  - Evaluates to value being assigned
- **Example**
  ```c
  int x = 1, y = 2, z = 3;
  x = (y = z);
  3 3 3
  ```
  evaluates to 3 (true)
  ```c
  if (x = 3) {
    ...
  }
  ```
  evaluates to 3
Complex Condition

• And
  \[
  \text{if } ((x > 0) \&\& (x \leq 10)) \quad 0 < x \leq 10
  \]
• Or
  \[
  \text{if } ((x > 10) \mid\mid (x < -10)) \mid |x| > 10
  \]
• Negation
  \[
  \text{if } (! (x > 0)) \quad \text{not} (x > 0) \iff x \leq 0
  \]

Beware that & and | are also C operators.

Lazy Logical Operator Evaluation

• If the conditions are sufficient to evaluate the entire expression, the evaluation terminates at that point => lazy

• Examples
  \[
  \text{if } ((x > 0) \&\& (x \leq 10)) \\
  \text{Terminates if } (x > 0)
  \]
  \[
  \text{if } ((x > 10) \&\& (x < 20)) \mid\mid (x < -10)) \\
  \text{Terminates if } (x > 10) \&\& (x < 20)
  \]

Use of Braces

\[
\text{if } (\text{choice} == 1) \{ \\
\quad \text{printf("1\n");}
\} \\
\text{else } \{ \\
\quad \text{printf("Other\n");}
\}
\]

When the operation is a single statement, '{' and '}' can be omitted.

switch Statement

\[
\text{switch } (\text{integer expression}) \{ \\
\quad \text{case constant: statements break;}
\} \\
\text{default: statements}
\]

Example

\[
\text{int } x, y, \text{ result } = 0; \text{ scanf("%d %d", \&x, \&y); }
\text{switch(x) \{} \\
\quad \text{case 1: break;}
\quad \text{case 2:}
\quad \text{ case 3: result } = 100; \\
\quad \text{ case 4:}
\quad \quad \text{switch(y) \{ }
\quad \quad \text{ case 5: result } += 200; \text{ break;}
\quad \quad \text{ default: result } = -200; \text{ break;}
\quad \text{ break;}
\quad \text{ default: result } = 400; \text{ break;}
\text{ \}}
\}
\]

break Fall Through

• Omitting \text{break} in a \text{switch} statement will cause program control to fall through to the next case
• Can be a very convenient feature
• Also generates very subtle bugs
• \text{switch} statements only test equality with integers
Section 5: while Loops

```c
while (true) {
    /* some operation */
}
```

while and Character Input

- **EOF** is a constant defined in stdio.h
  - Stands for End Of File

```c
int main() {
    int nc = 0, nl = 0; char c;
    while ((c = getchar()) != EOF) {
        nc++;
        if (c == '\n') nl++;
    }
    printf("Number of chars is %d and number of lines is %d\n", nc, nl);
    return 0;
}
```

Review: Assignment has value

- In C, assignment expression has a value, which is the value of the lefthand side after assignment.
- Parentheses in `(c = getchar()) != EOF` are necessary.
- `c = getchar() != EOF` is equivalent to `c = (getchar() != EOF)`
- `c` gets assigned 0 or 1.

Summary

- C and Java’s conditionals and loops are very similar
- C does not support booleans, uses 0 and 1 (not 0) instead
- Learn how to use `scanf` and `getchar`, especially with input loops
- Learn how C handles characters