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
- arithmetic
- assignment
- function parameters
- function return type
- promotion if possible

Explicit conversions
- casting

```
int x;
x = (int) 4.0;
```


Section 2

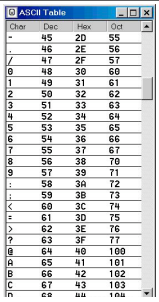
Use of `char` (character)

- 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

`char` Input/Output

- Input
  - **`char getchar()`** receives/returns a character
  - Built-in function
- Output
  - **`printf`** with **`%c`** specification

chartypes.c

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

Section 3

scanf Function

```
scanf("
  ",          );
```

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

scanf Function

- 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

scanf Function

```
int main() {
  int x;

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


scanf with multiple variables

scanf.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 Function

- Each variable in the list MUST be prefixed with an &.
- Read from standard input (the keyboard) and tries to match the input with the specified pattern, one by one.
- If successful, the variable is updated; otherwise, no change in the variable.
- The process stops as soon as **scanf** exhausts its format string, or matching fails.
- Returns the number of successful matches.

scanf Continued

- 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).

`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.

Section 4

`if-else` Statement

```
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;
}
```

menu.c

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)`** ≤

⚠ `==` (equality)
`=` (assignment)

The values are internally represented as integer.
true → 1 (not 0), false → 0

Assignment as Expression

- Assignment
  - Assignments are expressions
  - Evaluates to value being assigned
- Example

```
int x = 1, y = 2, z = 3;
x = (y = z);
```

3 ← 3 ← 3

evaluates to 3

evaluates to 3 (true)

```
if (x = 3) {
   ...
}
```

Complex Condition

- And

```
if ((x > 0) && (x <= 10))
```

$0 < x \le 10$

- Or

```
if ((x > 10) || (x < -10))
```

$|x| > 10$

- Negation

```
if (!(x > 0))
```

not $(x > 0) \Leftrightarrow x \le 0$

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
  - `if ((x > 0) && (x <= 10))`
    Terminates if `(x > 0)` fails
  - `if ((x > 10)&&(x < 20))||(x < -10))`
    Terminates if `(x > 10) && (x < 20)` succeeds

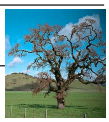
Use of Braces

```
if (choice == 1) {
  printf("1\n");
}
else {
  printf("Other\n");
}
```

```
if (choice == 1)
  printf("1\n");
else
  printf("Other\n");
```

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

`switch` Statement



```
switch (integer expression) {
case constant:
   statements
   break;
case constant:
   statements
   break;
possibly more cases
default:
   statements
}
```

Multi-branching

`break` Fall Through

- Omitting **`break`** in a **`switch`** statement will cause program control to fall through to the next case
- Can be a very convenient feature
- Also generates very subtle bugs
- **`switch`** statements only test equality with integers

Example

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

Section 5

`while` Loops

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


`while` and Character Input

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

`charloop.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.
- Parens 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