CS 50 - 1

Introduction

We can convert **source code** into **machine code** using a very special piece of software called **compiler**.

Code can be evaluated upon three axes.

- **Correctness**; refers to "does the code run as intended?"
- **Style**; refers to "how aesthetically pleasing and consistent is the code?"
- **Design**; refers to "how well is the code designed?"

VSCode

Visual studio code is a compiler with a lot of softwares pre- loaded on it.

We can build a program in C by typing `~$ code file-name.c` into the **terminal** window. Then, in the **text editor** that appears, you can write your cod

```
#include <stdio.h>
int main(void)
{
    printf(« hello, world\n »);
}
```

Clicking back in the terminal window, you can compile your code by executing `make filename`, Then type `./file_name` and your program will execute.

Functions

In C, to display any text on the screen, there is a function called printf.

```
printf("hello, world");
```

A common error in C programming is the omission of a semi-colon.

The argument passed to printf is "hello, world".

Variables

```
# include <stdio.h>

int main(void)
{
    string answer = get_string("what's your name ?");
    printf("hello, %s\n", answer);
}
```

answer is a special holding place we call a variable.

To bring a variable into existence, you need simply specify the data type of the variable and give it a name.

There re many data types, such as int, bool, char, etc...

If you wish to create multiple variables of the same type, you can specify the type name once, and then list as many of that type as you want.

```
int height, width;
float sqrt2, sqrt3, pi;
```

After a variable is declared, it's no longer necessary to specify it's type.

```
int number; //declaration
number = 17; // assignment
```

If you are simultaneously declaring and setting the value of a variable (sometimes called initializing), you can consolidate these to one step.

```
int number = 17; //initialization
char letter = 'N'; // initialization
```

Conditional

Conditional expressions allow your programs to make decisions and take different forks in the road, depending on the values of variables, or based on what the user inputs.

C provides a few different ways to implement conditional expressions (also known as **branches**) in programs.

```
if (boolean-expression) {
}
```

If the boolean-expression evaluates to true, then all the lines of code between the curly braces will execute in order from top to bottom. They will not, if the boolean-expression evaluate to false.

```
if (boolean-expression) {
} else {
}
```

If the boolean-expression evaluates to true, all the lines of code between the first set of curly braces will execute in order from top to bottom. And the lines of code between the second set of curly braces will execute, if it evaluates to false.

```
if (boolean-expr1) {      // first branch
} else if (boolean-expr2) { // second branch
} else if (boolean-expr3) { // third branch
} else { // fourth branch
}
```

In C it's possible to create an if-else if-else chain. Each branch is mutually exclusive.

It's also possible to create a chain of non-mutually exclusive branches.

```
if (boolean-exp1) { }  // first branch
if (boolean-exp2) { }  // second branch
if (boolean-exp3) {    // third branch
} else { // fourth branch
}
```

Switch() statement is a conditional statement that permits enumeration of discrete cases, instead of relying on boolean expressions.

It's important to break; between each cases, or you will "fall through" each case.

```
int x;
switch (x) {
    case 1:
        printf("One!\n");
        break;
    case 2:
        printf("One!\n");
        break;
    default:
        printf("One!\n");
        break;
}
```

Loops

Loops allow our program to execute lines of code repeatedly, saving you from having to copy paste or otherwise repeat lines of code.

```
while (true) {
}
```

This is what we call an infinite loop. These lines of code between the curly braces will execute repeatedly from top to bottom, until and unless we break out of it or otherwise kill our program.

If the boolean-expr evaluates to true all the lines of code between the curly braces will execute repeatedly, in order from top to bottom, until boolean-expr evaluates to false.

```
while (boolean-expr) {
}
```

```
do {
} while(boolean-expr);
```

The loop will execute all the lines of code between the curly brackets once, and if boolean-expr evaluates to true, will go back and repeat that process until boolean-expr evaluates to false.

Syntactically unintuitive, but for loops are used to repeat the body of a loop a specified number of times.

```
for( start; exp; increment) {
}
```

The statement(s) in start are executed.

The statement(s) in increment are executed.

The exp is checked
- If it evaluates to true, the body of the loop executes.
- If it evaluates to false, the body of the loop does not execute.

Loops give a better design to the code

While

Use when you want a loop to repeat an unknown number of times and possibly not at all.

do-while

Use when you want a loop to repeat an unknown number of times, but at least once.

for

Use when you want a loop to repeat a discrete number of times, though you may not know the number at the moment the program is compiled.

Operators

Operators refer to the mathematical operations that are supported by your compiler. In C, these mathematical operators include:

+ for addition
- for subtraction
* for multiplication
/ for division
% for remainder

C also provides a shorthand way to apply an arithmetic operator to a single variable.

```
x = x * 5;
x *= 5;
```

This trick works with all five basic arithmetic operators. C provides a further shorthand for incrementing or decrementing a variable by 1.

```
x++;
x--;
```

Boolean expressions

Boolean expressions are used in C for comparing values.

All boolean expressions in C evaluate one of two possible values - true or false.

We can use the result of evaluating a boolean expression in other programming constructs such as deciding which branch in a conditional to take, or determining whether a loop should continue to run.

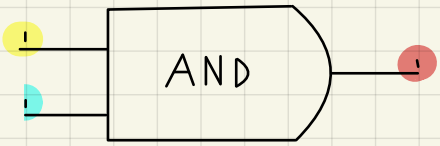
In C, every nonzero value is equivalent to true, and zero is false.

Two main types of Boolean expressions: logical operators and relational operators.

Logical Operators

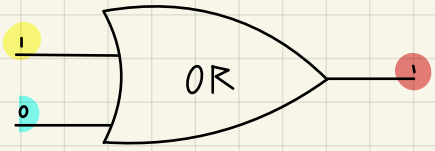
Logical AND (&&) is true if and only if both operands are true, otherwise false

X	Y	(X && Y)
true	true	true
true	false	false
false	true	false
false	false	false



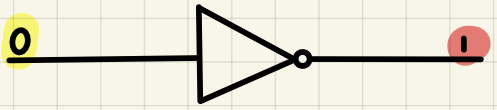
Logical OR (||) is true if and only if at least one operand is true, otherwise false.

X	Y	(X || Y)
true	true	true
true	false	true
false	true	true
false	false	false



Logical NOT (!) inverts the value of its operand.

X	!X
true	false
false	true



Relational Operators

They behave as you would expect them to, and appear syntactically similar to how you may recall them from elementary arithmetic.

- Less than (x < y)
- Less than or equal to (x <= y)
- Greater than (x > y)
- Greater than or equal to (x >= y)

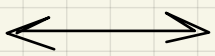
C can also test two variables for equality and inequality

- Equality (x == y)
- Inequality (x != y)

?:

The ternary operator (?:) is mostly a cute trick, but it's useful for writting trivially short conditional branches.

```
int x = (expr) ? 5 : 6;
```



```
int x;
if (expr) {
    x = 5;
} else {
    x = 6;
}
```

Comments

Comments are fundamental parts of computer program, where you leave explanatory remarks to yourself and others that may be collaborating with you regarding your code.

Typically each comment is a few words or more, providing the reader an opportunity to understand what is happening in a specific block of code. Further, such comments serve as a reminder for you later when you need to revise your code.

```
// print grid of bricks
for (int i=0; i<n; i++)
{
    for (int j=0; j<n; j++)
    {
        printf("#");
    }
    printf("\n");
}
```

Comments involve placing // into your code, followed by a comment.

Linux and the command Line

Linux is an operating system that is accessible via the command line in the terminal window of VSCode.

The following commands are working for any UNIX based system.

ls

Short for "list", this command will give you a readout of all the files and folders in the current directory

cd <directory>

short for "change directory"; this command change your current directory to <directory>, which you specify.

mkdir <directory>

short for "make directory"; this command will create a new subdirectory name <directory> located in the current directory.

cp <source> <destination>

Short for "copy", this command will allow you to create a duplicate of the file you specify as <source>, which it will save in <destination>.

rm <file>

Short for "remove", this command will delete <file> after it asks you to confirm (y/n) you want to delete it.

Abstraction

Abstraction is the art of simplifying our code such that it deals with smaller and smaller problems.

We can abstract away two problems into separate functions.

```
int get_size(void)
{
    ...
}
```

```
void print_grid(int n)
{
    ...
}
```

```
int main(void)
{
    int n = get_size();
    print_grid(n);
}
```

Data types

Types refer to the possible data that can be stored within a variable. Types are very important because each type has specific limits. Because of the limits in memory, the highest value of an int can be 4294967296.

bool	a Boolean expression of either true or false
char	a single character like a or 2
float	a floating-point value, or real number with a decimal value
double	a floating-point value with more digits than a float
int	integers up to a certain size, or number of bits
long	integers with more bits, so they can count higher than an int
string	a string of characters

Numbers	Text
int (%i)	char (%c)
float (%f)	string (%s)
long (%l)	
double (%f)	

– Types and format code

Int

Integers always take up 4 bytes of memory (32 bits).
This means the range of values they can store is necessarily limited to 32 bits worth of information

Integer Range:

-2^{31} ——— 0 ——— $2^{31}-1$

Void

Is a type but not a data type

Functions can have a void return type, which mean they don't return a value.

The parameter list of a function can be void. It means the function takes no parameter.

Think of void as a placeholder for "nothing."

Unsigned int

Unsigned is a qualifier that can be applied to certain types, which effectively doubles the positive range of variables of that type, at the cost of disallowing any negative values.

Unsigned Integer Range:

0 ——— 2^{31} ——— $2^{32}-1$

Float

Floating point values always take up
4 byte of memory (32 bits).

It's a little complicated to describe the range of float, but suffice it to say with 32 bits of precision, some of which might be used for an integer part, we are limited on how precise we can be.

Double

The difference is that doubles are double precision.
They always take up to 8 bytes of memory.

With an addition 32 bits of precision relative to a float, doubles allow us to specify much more precise real numbers.