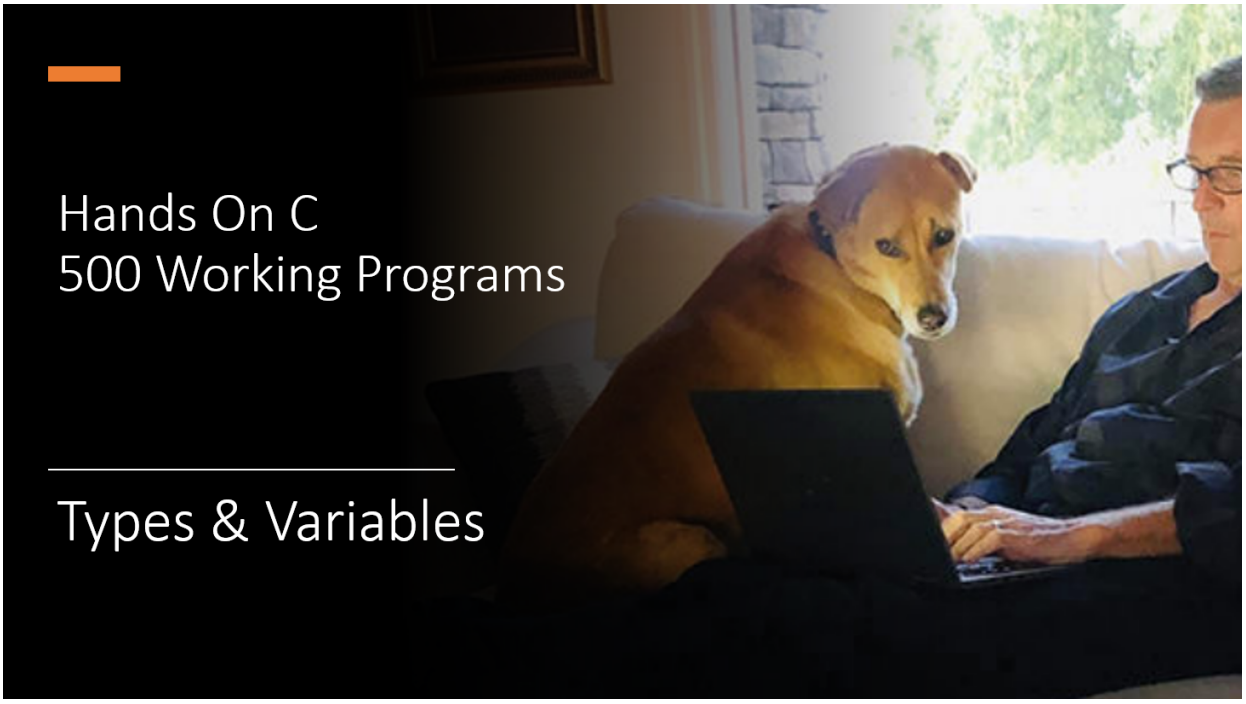

Programs Store Data within Variables

In [1]:
```c
#include <stdio.h>

int main(void)
{
   // Variable declarations go here

   printf("Hello, world");
}
```

```
Hello, world
```

In [2]:
```c
#include <stdio.h>

int main(void)
{
    int height; // student height
    int weight; // student weight
    int age;    // student age

    // Remaining program statements
}
```

Understanding C's Four Basic Variable Types

In [3]:
```c
#include <stdio.h>

int main(void)
{
   char letter;  // Stores a single character such as A through Z
   int count;    // Stores a counting number, either positive or negative
   float salary; // Stores a single-precision floating-point number such as 3.14 (
   double tax;   // Stores a double-precision floating-point number such as 1.234
}
```

Assigning a Value to a Variable

In [4]:
```c
#include <stdio.h>

int main(void)
{
    int height; // student height
    int weight; // student weight
    int age;    // student age

    height = 73;
    weight = 180;
    age = 30;

    // Remaining program statements
}
```

Declaring Multiple Variables of the Same Type on the Same Line

In [5]:
```
#include <stdio.h>

int main(void)
{
    int age;
    int weight;
    int height;
}
```

In [6]:
```
#include <stdio.h>

int main(void)
{
    int age, weight, height;
}
```

Commenting Your Variables at Declaration

In [7]:
```c
#include <stdio.h>

int main(void)
{
    int age;      // user's age
    int weight;   // user's weight in pounds
    int height;   // user's height in inches
}
```

Assigning Values to Variables at Declaration

In [8]:
```c
#include <stdio.h>

int main(void)
{
    int age = 30;      // user's age in years
    int height = 73;   // user's height in inches
    int weight = 180;  // user's weight in pounds
}
```

Initializing Same Line Variables at Declaration

In [9]:
```c
#include <stdio.h>

int main(void)
{
   int age = 30, weight = 180, height = 73;
}
```

Use Meaningful Variable Names

In [10]:
```c
#include <stdio.h>

int main(void)
{
    int a, b, c;   // not meaningful names

    int test_score, studentID, hoursWorked;  // better readable variable names
}
```

Understanding C's Keywords

```
auto     default     float     register     struct     volatile
break    do          for       return       switch     while
case     double      goto      short        typedef
char     else        if        signed       union
const    enum        int       sizeof       unsigned
continue extern      long      static       void
```

In [11]:
```c
#include <stdio.h>

int main(void)
{
   int switch;  // cannot use keyword switch for a variable name
}
```

```
/tmp/tmpgw85cvr7.c: In function ‘main’:
/tmp/tmpgw85cvr7.c:5:8: error: expected identifier or ‘(’ before ‘switch’
    int switch;  // cannot use keyword switch for a variable name
        ^~~~~~
[C kernel] GCC exited with code 1, the executable will not be executed
```

Understanding Variables of Type int

Stores positive and negative counting numbers -3, -2, -1, 0, 1, 2, 3

No decimal point

Value range is determined by the number of bytes the compiler uses to store a value of type int

Understanding Variables of Type char

```
Store a byte of data (-128 to 127)

Normally stores an ASCII character
```

Understanding Variables of Type float

```
A number with a decimal point such as 3.14

Normally has about seven digits of precision 1.23456789;
```

Understanding Variables of Type Double

```
A number with a decimal point

Twice the precision of a variable of type float (about 14 digits)

1.234567901234567;
```

Assigning Values to Floating-Point Variables

In [12]:
```c
#include <stdio.h>

int main(void)
{
    float pi = 3.14159;
    float small = 1.234e-5;
    double large = 1234567890.0;
}
```

Understanding Type Modifiers

In [13]:
```
#include <stdio.h>

int main(void)
{
    unsigned int currentInventory;
    register int loopCount;
    long int secondsSince1970;
}
```

Understanding the unsigned Type Modifier

```
Only stores positive numbers

Using the sign bit as part of the variable's value to increase the range of
values an integer can store
```

In [14]:
```
#include <stdio.h>

int main(void)
{
   unsigned int seconds_until_launch;     // will never be negative
   unsigned char extended_ASCII_code;     // ASCII characters 0 to 255
}
```

Understanding the Long Modifier

```
Stores very large numbers such as 123456789012345 as an integer value

When used with a constant, append an L 123456789012345L
```

In [15]:
```c
#include <stdio.h>

int main(void)
{
    long int national_debt = 30000000000000L;
}
```

Understanding the long unsigned Type Modifier

```
Only positive values

Uses the sign bit as part of the variable's value to increase the range of
values the variable can store
```

In [16]:
```
#include <stdio.h>

int main(void)
{
    unsigned long int current_time_in_seconds;
}
```

Working with Long Values

In [17]:
```c
#include <stdio.h>

int main(void)
{
    long int million = 1000000L;       // Note L at end of constant
    long int big_number = 1234567890L;
}
```

Understanding the register Type Modifier

In [18]:
```c
#include <stdio.h>

int main(void)
{
  register int count;
  register unsigned timer;
}
```

Understanding the short Type Modifier

In [19]:
```c
#include <stdio.h>

int main(void)
{
   short int count;   // Typically -32,768 through 32,767
}
```

Omitting int from Modified Declarations

In [20]:
```c
#include <stdio.h>

int main(void)
{
    short int year = 2021;
    short last_year = 2020;

    unsigned int seconds;
    unsigned hours;
}
```

Understanding the signed Type Modifier

In [21]:
```c
#include <stdio.h>

int main(void)
{
   signed char ASCII_character;
   signed int seconds;
}
```

Multiple Assignment Operations

In [22]:
```c
#include <stdio.h>

int main(void)
{
   int counter = 0;
   int sum = 0;
   int value = 0;

   int Count, Sum, Value = 0;   // assign each variable the value 0
}
```

Assigning the Value of One Type of Variable to Another

In [23]:
```c
#include <stdio.h>

int main(void)
{
    int value;
    float pi = 3.14159;
    float cost = 9.95;

    value = pi;
    printf("Value %d\n", value);

    value = cost;
    printf("Value %d\n", value);
}
```

```
Value 3
Value 9
```

Creating Your Own Data Types

In [24]:
```c
#include <stdio.h>

int main(void)
{
    unsigned long int secondsSince1970;
    unsigned long int population;

    typedef unsigned long int ULINT;
    ULINT distanceToMoon;
}
```

Assigning an Octal or Hexadecimal Value

In [25]:
```c
#include <stdio.h>

int main(void)
{
    int octalValue = 0123;
    int hexValue = 0xA3FF;
}
```

Understanding Overflow

```
0          0000 0000 0000 0000
1          0000 0000 0000 0001
2          0000 0000 0000 0010
3          0000 0000 0000 0011

32,765     0111 1111 1111 1101
32,766     0111 1111 1111 1110
32,767     0111 1111 1111 1111

+1

-32-768    1111 1111 1111 1111
```

In [26]:
```c
#include <stdio.h>

int main(void)
{
    short int positive = 32767;
    printf("Positive %d\n", positive);

    positive = positive + 1;
    printf("Positive %d\n", positive);
}
```

```
Positive 32767
Positive -32768
```

Understanding Precision

In [27]:
```c
#include <stdio.h>

int main(void)
{
    float accurateValue = 0.12345678901234567890;
    double moreAccurate = 0.12345678901234567890;

    printf("accurateValue %21.20f\n", accurateValue);
    printf("moreAccurate %21.20f", moreAccurate);
}
```

```
accurateValue 0.12345679104328155518
moreAccurate 0.12345678901234567737
```

Understanding Enumerated Types

In [28]:
```
#include <stdio.h>

int main(void)
{
  enum sports { basketball, baseball, football, soccer } game;

  game = basketball;

  switch (game)
  {
    case basketball: printf("Favorite player Michael Jordan\n");
                     break;
    case football: printf("Favorite player Troy Aikmen\n");
                   break;
    case baseball: printf("Favorite player Willie Mays\n");
                   break;
    case soccer: printf("Favorite play Pele\n");
                 break;
  }
}
```

```
Favorite player Michael Jordan
```

Looping Through an Enumerated List

In [29]:
```
#include <stdio.h>

int main(void)
{
  enum sports { basketball, baseball, football, soccer } game;

  for (game = basketball; game <= soccer; game++)
    switch (game)
    {
      case basketball: printf("Favorite player Michael Jordan\n");
                       break;
      case football: printf("Favorite player Troy Aikmen\n");
                     break;
      case baseball: printf("Favorite player Willie Mays\n");
                     break;
      case soccer: printf("Favorite play Pele\n");
                   break;
  }
}
```

```
Favorite player Michael Jordan
Favorite player Willie Mays
Favorite player Troy Aikmen
Favorite play Pele
```

Understanding Enumerated List Values

In [30]:
```c
#include <stdio.h>

int main(void)
{
  enum sports { basketball, baseball, football, soccer } game;

  printf("basketball %d baseball %d football %d soccer %d\n",
    basketball, baseball, football, soccer);
}
```

```
basketball 0 baseball 1 football 2 soccer 3
```

Assigning Specific Values to Enumerated Types

In [31]:
```c
#include <stdio.h>

int main(void)
{
  enum sports { basketball = 10, baseball = 20, football = 30, soccer = 40 } game

  printf("basketball %d baseball %d football %d soccer %d\n",
    basketball, baseball, football, soccer);
}
```

```
basketball 10 baseball 20 football 30 soccer 40
```

What You will Learn Next

```
C programs make extensive use of the printf function to display output.

printf("%d %f %s\n", 1, 3.14, "Hello, world");
```