Lesson 3

Memory, Variables, and Constants
**RAM** = Random Access Memory

- Each memory location stores 1 Byte = 8 bits
- The number of bits in the address determines how much RAM can be accessed
- Carefully distinguish between the address and the content of a RAM location!

<table>
<thead>
<tr>
<th>Address</th>
<th>Contents</th>
</tr>
</thead>
<tbody>
<tr>
<td>00000000</td>
<td>11100011</td>
</tr>
<tr>
<td>00000001</td>
<td>10101001</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td>11111100</td>
<td>00000000</td>
</tr>
<tr>
<td>11111101</td>
<td>11111111</td>
</tr>
<tr>
<td>11111110</td>
<td>10101010</td>
</tr>
<tr>
<td>11111111</td>
<td>00110011</td>
</tr>
</tbody>
</table>
QUIZ

• How many Bytes are in the memory pictured here?

• What is stored at address 1111 1101?
QUIZ

• How many Bytes are in the memory pictured here?

The size of the memory is 256 B.

The content of address 1111 1101 is 1111 1111.
QUIZ

• How many Bytes are in a memory whose addresses have 10 bits?
QUIZ

• How many Bytes are in a memory whose addresses have 10 bits?

The size of the memory is 1024 B = 1 KB.
Binary vs. decimal multipliers

$2^{10} = 1024 \approx 1,000$

$2^{20} = 1024 \times 1024 \approx 1,000,000$

$2^{30} = 1024 \times 1024 \times 1024 \approx 1,000,000,000$
QUIZ

• How many Bytes are in a memory whose addresses have 24 bits?
QUIZ

• How many Bytes are in a memory whose addresses have 16 bits?

The size of the memory is 16 MB.
QUIZ

• Why were the previous-generation desktop computers limited to 4 GB of memory?

Hint: They were “32-bit machines”, in particular memory addresses were represented on 32 bits.
QUIZ

• What is the memory limitation of today’s 64-bit desktops?
<table>
<thead>
<tr>
<th>Variable Type</th>
<th>Keyword</th>
<th>Bytes Required</th>
<th>Range</th>
</tr>
</thead>
<tbody>
<tr>
<td>Character</td>
<td>char</td>
<td>1</td>
<td>-128 to 127</td>
</tr>
<tr>
<td>Short integer</td>
<td>short</td>
<td>2</td>
<td>-32767 to 32767</td>
</tr>
<tr>
<td>Integer</td>
<td>int</td>
<td>4</td>
<td>-2,147,483,647 to 2,147,438,647</td>
</tr>
<tr>
<td>Long integer</td>
<td>long</td>
<td>4</td>
<td>-2,147,483,647 to 2,147,438,647</td>
</tr>
<tr>
<td>Unsigned character</td>
<td>unsigned char</td>
<td>1</td>
<td>0 to 255</td>
</tr>
<tr>
<td>Unsigned short integer</td>
<td>unsigned short</td>
<td>2</td>
<td>0 to 65535</td>
</tr>
<tr>
<td>Unsigned integer</td>
<td>unsigned int</td>
<td>4</td>
<td>0 to 4,294,967,295</td>
</tr>
<tr>
<td>Unsigned long integer</td>
<td>unsigned long</td>
<td>4</td>
<td>0 to 4,294,967,295</td>
</tr>
<tr>
<td>Unsigned long long integer</td>
<td>unsigned long long</td>
<td>8</td>
<td>0 to 18,446,744,073,709,551,615</td>
</tr>
<tr>
<td>Single-precision</td>
<td>float</td>
<td>4</td>
<td>1.2E-38 to 3.4E38</td>
</tr>
<tr>
<td>floating-point</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Double-precision</td>
<td>double</td>
<td>8</td>
<td>2.2E-308 to 1.8E308</td>
</tr>
</tbody>
</table>

1. Approximate range; precision = 7 digits.
2. Approximate range; precision = 19 digits.
/* sizeof.c -- Program to tell the size of the C variable */

#include <stdio.h>

int main(void)
{
    printf("\nA char is \%d bytes", sizeof(char));
    printf("\nA int is \%d bytes", sizeof(int));
    printf("\nA short is \%d bytes", sizeof(short));
    printf("\nA long is \%d bytes", sizeof(long));
    printf("\nA long long is \%d bytes\n", sizeof(long long));
    printf("\nA unsigned char is \%d bytes", sizeof(unsigned char));
    printf("\nA unsigned int is \%d bytes", sizeof(unsigned int));
    printf("\nA unsigned short is \%d bytes", sizeof(unsigned short));
    printf("\nA unsigned long is \%d bytes", sizeof(unsigned long));
    printf("\nA unsigned long long is \%d bytes\n", sizeof(unsigned long long));
    printf("\nA float is \%d bytes", sizeof(float));
    printf("\nA double is \%d bytes\n", sizeof(double));
    printf("\nA long double is \%d bytes\n", sizeof(long double));
    return 0;
}
Output ▼

<table>
<thead>
<tr>
<th>Type</th>
<th>Size</th>
</tr>
</thead>
<tbody>
<tr>
<td>A char</td>
<td>1 byte</td>
</tr>
<tr>
<td>An int</td>
<td>4 bytes</td>
</tr>
<tr>
<td>A short</td>
<td>2 bytes</td>
</tr>
<tr>
<td>A long</td>
<td>4 bytes</td>
</tr>
<tr>
<td>A long long</td>
<td>8 bytes</td>
</tr>
<tr>
<td>An unsigned char</td>
<td>1 byte</td>
</tr>
<tr>
<td>An unsigned int</td>
<td>4 bytes</td>
</tr>
<tr>
<td>An unsigned short</td>
<td>2 bytes</td>
</tr>
<tr>
<td>An unsigned long</td>
<td>4 bytes</td>
</tr>
<tr>
<td>An unsigned long long</td>
<td>8 bytes</td>
</tr>
<tr>
<td>A float</td>
<td>4 bytes</td>
</tr>
<tr>
<td>A double</td>
<td>8 bytes</td>
</tr>
<tr>
<td>A long double</td>
<td>12 bytes</td>
</tr>
<tr>
<td>Full Name</td>
<td>Commonly Used Keyword</td>
</tr>
<tr>
<td>-------------------</td>
<td>-------------------------------</td>
</tr>
<tr>
<td>char</td>
<td>signed char</td>
</tr>
<tr>
<td>short</td>
<td>signed short int</td>
</tr>
<tr>
<td>int</td>
<td>signed int</td>
</tr>
<tr>
<td>long</td>
<td>signed long int</td>
</tr>
<tr>
<td>long long</td>
<td>signed long long int</td>
</tr>
<tr>
<td>unsigned char</td>
<td>unsigned char</td>
</tr>
<tr>
<td>unsigned short</td>
<td>unsigned short int</td>
</tr>
<tr>
<td>unsigned int</td>
<td>unsigned int</td>
</tr>
<tr>
<td>unsigned long</td>
<td>unsigned long int</td>
</tr>
<tr>
<td>unsigned long long</td>
<td>unsigned long long int</td>
</tr>
</tbody>
</table>
How do we declare and initialize variables?

```c
#include <stdio.h>

void main()
{
    unsigned u1 = 42, u2;
    int i1 = -42, i2;

    u2 = 43;
    i2 = -43;

    printf("%u %u \n", u1, u2);
    printf("%d %i \n", i1, i2);
}
```
How do we print long long variables?

```c
#include <stdio.h>

int main(void)
{
    unsigned big = 4000000000;
    unsigned too_big = 4300000000;
    unsigned long long huge = 10000000000;

    printf("Unsigned long integer: %lu \n", big);
    printf("Unsigned long integer overflow: %lu \n\n", too_big);
    printf("Unsigned long long integer: %lld \n\n", huge);
}
```

Output:

```
Unsigned long integer: 4000000000
Unsigned long integer overflow: 5032704
Unsigned long long integer: 10000000000
```
QUIZ

Is it possible to overflow an unsigned long long int?
QUIZ

Is it possible to overflow an unsigned long long int?

<table>
<thead>
<tr>
<th>Unsigned long long</th>
<th>unsigned long</th>
<th>8</th>
<th>0 to 18,446,744,073,709,551,615</th>
</tr>
</thead>
<tbody>
<tr>
<td>integer</td>
<td>long</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
To do for next time:

• Read the first part of Lesson 3 up to p.45 and take notes in the notebook.

• Answer Quiz questions 1, 2, 3 in the notebook.
QUIZ

Write a C statement (one line) that displays the size (in Bytes) of short integers and long long integers on the local machine.
QUIZ

Convert 42 to binary with pencil and paper.
/* Demonstrates variables and constants */
#include <stdio.h>

/* Define a constant to convert a number of laps to miles */
#define LAPS_PER_MILE 4

/* Define a constant for the current year */
const int CURRENT_YEAR = 2013;

/* Declare the needed variables */
float miles_covered;
int laps_run, year_of_birth, current_age;

int main( void )
{
    /* Input data from user */

    printf("How many laps did you run: ");
    scanf("%d", &laps_run);
    printf("Enter your year of birth: ");
    scanf("%d", &year_of_birth);

    /* Perform conversions */

    miles_covered = (float)laps_run/LAPS_PER_MILE;
    current_age = CURRENT_YEAR - year_of_birth;

    /* Display results on the screen */

    printf("You ran %.2f miles.\n", miles_covered);
    printf("Not bad for someone turning %d this year!\n", current_age);

    return 0;
}
/* Demonstrates variables and constants */
#include <stdio.h>

/* Define a constant to convert a number of laps to miles */
#define LAPS_PER_MILE 4

/* Define a constant for the current year */
const int CURRENT_YEAR = 2013;

/* Declare the needed variables */
float miles_covered;
int laps_run, year_of_birth, current_age;
int main( void )
{
    /* Input data from user */

    printf("How many laps did you run: ");
    scanf("%d", &laps_run);
    printf("Enter your year of birth: ");
    scanf("%d", &year_of_birth);

    /* Perform conversions */

    miles_covered = (float)laps_run/LAPS_PER_MILE;
    current_age = CURRENT_YEAR - year_of_birth;

    /* Display results on the screen */

    printf("\nYou ran %.2f miles.", miles_covered);
    printf("\nNot bad for someone turning %d this year!\n", current_age);

    return 0;
}
Integer division

There are 9 children at a party, and we have 24 cookies. How do we distribute the cookies equally?
Integer division

24 / 9 = 2 rem.6

24 / 9 \rightarrow 2 \quad \text{quotient}
24 \% 9 \rightarrow 6 \quad \text{remainder}
QUIZ  Integer division

24 / 9 = 2 rem.6

15 / 7 =
21 / 4 =
27 / 6 =
1 / 3 =
Integer division

miles_covered = 17 / 4;
miles_covered = (float) 17 / 4;
miles_covered = (float) (17 / 4);
Why is `sizeof` an operator, and not a function?

Reason 1: It’s evaluated at compile time, not execution time!
Why is `sizeof` an operator, and not a function?

Reason 2: Functions take numeric data as arguments, not types!
SKIP the `typedef` keyword (p.45)
Literal constants vs. symbolic constants

#define PI 3.1415  // preprocessor directive

int count = 20;
const float tax_rate = 0.25;
Preprocessor

- Extra step between Edit and Compile!
- This is where **directives** (like `#include` and `#define`) are processed.
To do in notebook for next time

Answer all the end-of-chapter questions:

- Q&A
- Workshop
- Quiz
- Exercises