Addition program

```c
#include <stdio.h>

int main()
{
    int integer1; /* first number to be input by user */
    int integer2; /* second number to be input by user */
    int sum;      /* variable in which sum will be stored */

    printf( "Enter first integer\n" ); /* prompt */
    scanf( "%d", &integer1 );          /* read an integer */

    printf( "Enter second integer\n" ); /* prompt */
    scanf( "%d", &integer2 );          /* read an integer */

    sum = integer1 + integer2;          /* assign total to sum */

    printf( "Sum is %d\n", sum );       /* print sum */

    return 0; /* indicate that program ended successfully */
}
```

Enter first integer
45
Enter second integer
72
Sum is 117
2.3 Another Simple C Program: Adding Two Integers

• As before
  – Comments, `#include <stdio.h>` and `main`

• `int integer1, integer2, sum;`
  – Definition of variables
    • Variables: locations in memory where a value can be stored
    • `int` means the variables can hold integers (-1, 3, 0, 47)
  – Variable names (identifiers)
    • `integer1, integer2, sum`
    • Identifiers: consist of letters, digits (cannot begin with a digit) and underscores(_)
      – Case sensitive
  – Definitions appear before executable statements
    • If an executable statement references and undeclared variable it will produce a syntax (compiler) error

variable names

• Valid names
  – a
  – sum
  – Sum
  – grade1
  – mis_student

• Invalid names
  – 1a
  – S um
  – 1grade
  – mis student
  – mis-student
2.3 Another Simple C Program: Adding Two Integers

- `scanf( "%d", &integer1 );`
  - Obtains a value from the user
    - `scanf` uses standard input (usually keyboard)
  - This `scanf` statement has two arguments
    - `%d` - indicates data should be a decimal integer
    - `&integer1` - location in memory to store variable
    - `&` is confusing in beginning – for now, just remember to include it with the variable name in `scanf` statements
  - When executing the program the user responds to the `scanf` statement by typing in a number, then pressing the `enter` (return) key

- `==` (assignment operator)
  - Assigns a value to a variable
  - Is a binary operator (has two operands)
  - `sum = variable1 + variable2;`
    - `sum` gets `variable1 + variable2`;
  - Variable receiving value on left

- `printf( "Sum is %d\n", sum );`
  - Similar to `scanf`
    - `%d` means decimal integer will be printed
    - `sum` specifies what integer will be printed
2.4 Memory Concepts

• Variables
  – Variable names correspond to locations in the computer’s memory
  – Every variable has a name, a type, a size and a value
  – Whenever a new value is placed into a variable (through `scanf`, for example), it replaces (and destroys) the previous value
  – Reading variables from memory does not change them

• A visual representation

```
integer1  45
```

```
integer1  45
integer2  72
sum       117
```
2.5 Arithmetic

- Arithmetic calculations
  - Use * for multiplication and / for division
  - Integer division truncates remainder
    • 7 / 5 evaluates to 1
  - Modulus operator(%) returns the remainder
    • 7 % 5 evaluates to 2

- Operator precedence
  - Some arithmetic operators act before others (i.e., multiplication before addition)
    • Use parenthesis when needed
  - Example: Find the average of three variables a, b and c
    • Do not use: a + b + c / 3
    • Use: (a + b + c) / 3

### Arithmetic operators:

<table>
<thead>
<tr>
<th>C operation</th>
<th>Arithmetic operator</th>
<th>Algebraic expression</th>
<th>C expression</th>
</tr>
</thead>
<tbody>
<tr>
<td>Addition</td>
<td>+</td>
<td>f + 7</td>
<td>f + 7</td>
</tr>
<tr>
<td>Subtraction</td>
<td>-</td>
<td>p - c</td>
<td>p - c</td>
</tr>
<tr>
<td>Multiplication</td>
<td>*</td>
<td>bm</td>
<td>b * m</td>
</tr>
<tr>
<td>Division</td>
<td>/</td>
<td>x / y</td>
<td>x / y</td>
</tr>
<tr>
<td>Modulus</td>
<td>%</td>
<td>r mod s</td>
<td>r % s</td>
</tr>
</tbody>
</table>

### Rules of operator precedence:

<table>
<thead>
<tr>
<th>Operator(s)</th>
<th>Operation(s)</th>
<th>Order of evaluation (precedence)</th>
</tr>
</thead>
<tbody>
<tr>
<td>()</td>
<td>Parentheses</td>
<td>Evaluated first. If the parentheses are nested, the expression in the innermost pair is evaluated first. If there are several pairs of parentheses “on the same level” (i.e., not nested), they are evaluated left to right.</td>
</tr>
<tr>
<td>* / %</td>
<td>Multiplication, Division, Modulus</td>
<td>Evaluated second. If there are several, they are evaluated left to right.</td>
</tr>
<tr>
<td>+ -</td>
<td>Addition, Subtraction</td>
<td>Evaluated last. If there are several, they are evaluated left to right.</td>
</tr>
</tbody>
</table>
2.6 Decision Making: Equality and Relational Operators

Step 1. \( y = 2 \times 5 \times 5 + 3 \times 5 + 7; \) (Leftmost multiplication)

- \( 2 \times 5 \) is 10

Step 2. \( y = 10 \times 5 + 3 \times 5 + 7; \) (Leftmost multiplication)

- \( 10 \times 5 \) is 50

Step 3. \( y = 50 + 3 \times 5 + 7; \) (Multiplication before addition)

- \( 3 \times 5 \) is 15

Step 4. \( y = 50 + 15 + 7; \) (Leftmost addition)

- \( 50 + 15 \) is 65

Step 5. \( y = 65 + 7; \) (Last addition)

- \( 65 + 7 \) is 72

Step 6. \( y = 72; \) (Last operation—place 72 in \( y \))

Examples of assignments

```c
int a, b, sum;
a = 2;
b = 3;
sum = a + b;
/* it is possible to compute value of sum as 
a and b are defined as integers whose values are known */
*/
```
Examples of assignments

```c
int a, b, sum;
a = 2;
sum = a + b;
/* it is not possible to compute value of `sum` as
   `a` and `b` are defined as integers but value of `b` is
   not known
*/
```

Examples of assignments

```c
int a, sum;
a = 2;
sum = a + b;
/* it is not possible to compute value of `sum` as
   `a` and `b` is defined as integers but `b` is not
   `b` appears first in `sum = a+b`
   its type is not known
   no space is allocated as a variable
   its value is not known
*/
```
int a, b, sum;
a = 2;
b = 3;
sum = a; // sum’s value is 2
sum = sum + b;
/* sum can be computed in two steps as
   in second statement
   right hand side is 2 + 3 which is 5
   5 is assigned to sum again its first value of 2 is lost

Two decimal types
- float and double for storing decimal numbers
- double memory capacity is higher than float
- Large numbers or more precisions can be stored
Example

- #include <stdio.h>
- int main()
- {
-   double area, hight,width;
-   printf("Enter hight:");
-   scanf("%lf",&hight), /* get value from the user and store in hight */
-   printf("Enter width:");
-   scanf("%lf",&width), /* get value from the user and store in with */
-   area = hight*width;
-   printf("area is %f\n",area);
-   return 0;
- }

Example without area variable

- #include <stdio.h>
- int main()
- {
-   double hight,width;
-   printf("Enter hight:");
-   scanf("%lf",&hight), /* get value from the user and store in hight */
-   printf("Enter width:");
-   scanf("%lf",&with), /* get value from the user and store in with */
-   // printing without an area variable
-   printf("area is %f\n",hight*width);
-   return 0;
- }
Double or float

• The hight, width and area variables can be declared as float as well

• The result is the same but when reading the values from the user use
  – scanf("%f",&hight);
  – scanf("%f",&width);

• Since memory is cheap and available, in our applications we mostly use double for decimal numbers

Format specifiers for double and float

• Format specifiers for double variables
• %lf for reading by scanf
• %f for writing by printf
• e.g. scanf("%lf",&hight);
• printf("area is %f \n",area);
• Format specifiers for float variables
• %f for reading by scanf
• %f for writing by printf
## Format specifiers for simple variable types

<table>
<thead>
<tr>
<th>Data types</th>
<th>printf conversion specifications</th>
<th>scanf conversion specifications</th>
</tr>
</thead>
<tbody>
<tr>
<td>long double</td>
<td>%Lf</td>
<td>%Lf</td>
</tr>
<tr>
<td>double</td>
<td>%f</td>
<td>%lf</td>
</tr>
<tr>
<td>float</td>
<td>%f</td>
<td>%f</td>
</tr>
<tr>
<td>unsigned long int</td>
<td>%lu</td>
<td>%lu</td>
</tr>
<tr>
<td>long int</td>
<td>%ld</td>
<td>%ld</td>
</tr>
<tr>
<td>unsigned int</td>
<td>%u</td>
<td>%u</td>
</tr>
<tr>
<td>int</td>
<td>%d</td>
<td>%d</td>
</tr>
<tr>
<td>unsigned short</td>
<td>%hu</td>
<td>%hu</td>
</tr>
<tr>
<td>short</td>
<td>%hd</td>
<td>%hd</td>
</tr>
<tr>
<td>char</td>
<td>%c</td>
<td>%c</td>
</tr>
</tbody>
</table>