Declaring a place to store numbers/values

• Use a variable declaration
• Variable declarations include
  • Variable type
  • Variable name
  • Optional initial value
  • Semicolon
• Examples:
  • int altitude=0;
  • char letter=´a´;
  • float temperature=1.1;
  • double force=1.2;
Arithmetic Expressions

- Perform basic computation

```c
int i1 = 1;    /* first integer */
int i2 = 2;    /* second integer */
int sum;       /* result */
sum = i1 + i2; /* compute sum */
```

- Adds the value in `i1` to the value in `i2`
  - Gets the number 3
  - Stores this value (3) in `sum`
Arithmetic Expressions

• Consider:

```c
int i = 1; /* integer */
i = i + 1;
```

• Does **NOT** mean the same thing as it does algebra
• Take the value in `i`
• Add 1 to it
• Store this new value in `i`
Arithmetic Expressions

• Can do other operations
• For example
  \[ x = y \times 3; \]
  \[ z = (a + 2) - b; \]
• Basic form
• variable to store result in = arithmetic expression;
• Semicolon tells C compiler that statement is finished
Printing Values to Screen

- Use the printf statement
- To use printf statement type "#include<stdio.h>" at the top of your C file
- Basic form:
  - printf("formatting string", argument1, argument2...);
- Formatting string includes
  - text to print
  - %d - for decimal integer arguments
  - %f – for floating point arguments
  - many others – look in book
  - in the order that the arguments are listed
Printing Values to Screen

- `printf("formatting string", argument1, argument2...);`
- Formatting string also includes escape codes
  - `\n` prints a new line
  - `\` prints a `\` character
  - `"` prints a `"` character
- `;` at the end of the printf statement tells the compiler that the statement is finished!!!
Printing Values to Screen

• Example:

  int x=1;
  printf("The value of x is %d.\n",x);

  float g=1.2;
  printf("The value of g is %f.\n",g);
Getting Input

• Data input using `scanf()` function

• Need to type “#include <stdio.h>” at the top of your C file to use the scanf function

• Basic form:
  ```c
  scanf("formatting string", &argument);
  ```

• Formatting string includes
  • `%d` - for decimal integer arguments
  • `%f` - for floating point arguments
  • many others - look in book

• Need semicolon at end!!!
Getting Input

`scanf("formatting string", &argument);`

- Why do we have the `&`?
- If we call a function on a variable (i.e. `function(var)`) we pass the value in var to the function
- Doesn’t give scanf a way to return a value
- `&argument` passes the location of argument to scanf
- Similar to giving someone your street address - they can use the address to go to your place and write down a message
Comments

- Use /* to start a comment, and
- */ to end the comment
- The compiler ignores whatever is in between
- Comments can span multiple lines
- Caveat – it doesn’t ignore */ so do not include that in a comment
Main Function

• C program start & end w/ main function
• Function declaration of form
• int main(void)
• int means that the function main returns an integer value
• main is the functions name
• void means that the function doesn’t take any input
Main Function

• Then we have a `{` to indicate that the function body starts
• Function body starts with variable declarations
• Then it gives the sequence of operations the function performs (as we’ve described already)
• Ends with a `}`
Boolean Expressions

• **Actually integers in C**
  • 0 = false
  • Other integers = true

• Typically use comparison operations
  • For example:
    - a>b - true if a>b
    - a>=b - true if a>=b
    - a<b - true if a<b
    - a<=b - true if a<=b
    - a==b - true if a equals b
    - a!=b - true if a isn’t equal to b

• CAVEAT: BE WARY OF a=b
  • This assigns a to be the value of b and then returns this value
  • Common typo/mistake
More Boolean Expressions

- **Negation**
  - \(!a\) – true if \(a\) is false

- **OR**
  - \(a||b\) – true if \(a\) or \(b\) is true

- **AND**
  - \(a&&b\) – true if \(a\) and \(b\) is true
Control Flow

- If/else statements same as Java
- Loop constructs (while, do while, for) also same
- Break/continue same
- Switch constructs same
- Goto statements supported – use with care
Arithmetic Operations in C

- Arithmetic Operators
  - parentheses
  - unary plus, minus
  - multiplication, division, modulo
  - addition, subtraction
  - shift left, shift right

- Evaluation order of expressions
  - usually left to right
  - by operator precedence
    - ordered as in table above (higher operators are evaluated first)

- Arithmetic operators are available
  - for integer types: all
  - for floating point types: all except \( \% \), \( <<, >> \)