

Variables

- A variable is a reserved **location in memory** that
 - has a **name**
 - has an associated **type** (for example, integer)
 - holds **data** which can be **modified**

Variables

- In order to use a variable in our program we must first **declare** it.
- HOW?
 - A declaration statement has the format:

```
type variable_name;
```

- **type** : *what kind of data will be stored in that location (integer? character? floating point?)*
- **variable_name** : *what is the name of the variable?*
- **semi-colon** : *indicates this is a statement!*

Variable types

- There are four basic data types in C

<u>Type</u>	<u>C keyword to use:</u>
Integer	<i>int</i>
Floating point	<i>float</i> <i>double</i>
Character	<i>char</i>

Variable types

- **int**
 - Integer variables hold **signed whole numbers** (i.e. with no fractional part), such as 10, - 4353, etc.
 - Integers typically take up 4 bytes (= 32 bits, 1 for the sign, 31 for the number).
 - The range of integers is typically
 - from - 2^{31} (approx -10^9) to + 2^{31} (approx. 10^9)

Variable types

- **float** and **double**

- floating point variables hold **signed floating point numbers** (i.e. with a fractional part), such as 10.432, - 33.335, etc.
- double provides twice the precision of float.
- floats typically take up 4 bytes
- doubles take up 8 bytes
- The range of floats is approximately $\pm 2^{127}$ ($\pm 10^{38}$)
- The range of doubles is approximately $\pm 2^{1023}$ ($\pm 10^{308}$)

Variable types

- **char**

- character variables hold **single characters**, such as 'a', '\n', ' ', etc.
- characters usually take 1 byte (8 bits).
- **IMPORTANT** : Note that **the value of a character is enclosed in single quotes**.
- Each character is essentially "encoded" as an integer.
 - A computer normally stores characters using the ASCII code (American Standard Code for Information Exchange)

Variable types

- **char** (continued)

- ASCII is used to represent
 - the characters A to Z (both upper and lower case)
 - the digits 0 to 9
 - special characters (e.g. @, <, etc)
 - special control codes
- For example,
 - the character 'A' is represented by the code 65
 - the character '1' is represented by the code 49
 - **IMPORTANT: the integer 1, the character '1' and the ASCII code 1 represent three different things!**

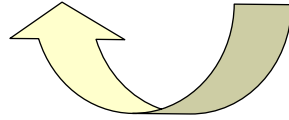
Variable values

- After a variable has been declared, its memory location contains randomly set bits. In other words, it does not contain **valid data**.
- The **value** stored in a variable **must be initialized** before we can use it in any computations.
- There are two ways to initialize a variable:
 - by assigning a value using **an assignment statement**
 - by reading its value from the keyboard

Variable values

- The basic syntax of an assignment statement is

- **variable = value;**



assign the value on the right hand side
to the variable on the left hand side

- Example

```
int num_students; // declare
num_students = 22; // initialize
```

Literals

- Literals are fixed values written into a program.

- Example:

```
char keypressed;
keypressed = 'y'; /* 'y' is a character literal */
```

- Example:

```
double pi;
pi = 3.14; /* 3.14 is a floating-point literal. */
```

- Example:

```
int index;
index = 17; /* 17 is an integer literal */
```

Example

```
/* sample program that demonstrates  
variable declaration and  
initialization. */
```

```
#include <stdio.h>
```

```
int main ()  
{  
    int num_students;  
    num_students = 22;  
    return 0;  
}
```

Example

```
/* sample program that demonstrates variable  
declaration and initialization. */
```

```
#include <stdio.h>
```

```
int main ()  
{  
    double rate, amount; /* declare two  
                           double variables */  
    amount = 12.50;  
    rate = 0.05;  
    return 0;  
}
```

Example

```
/* sample program that demonstrates how to
   declare and initialize a variable at
   the same time */

#include <stdio.h>

int main ()
{
    char grade = 'A';
    return 0;
}
```

Example

```
/* sample program that demonstrates how to
   declare and initialize a variable at
   the same time */

#include <stdio.h>

int main ()
{
    char pass_grade = 'A', fail_grade = 'F';
    return 0;
}
```

Printing variable values

- `printf()` will print formatted output to the screen.

- To print a message:

```
printf("This is a message\n");
```

this is a string literal

- How do we print the value of a variable?
- Answer: Use special **format specifiers** depending on the type of the variable

Printing variable values

- To print an integer:

```
int degreesF = 68;  
printf("The temperature is %d degrees.", degreesF);
```

Specifier for
"print an integer value"

"Read value from
this variable"

Output:

```
> The temperature is 68 degrees.
```

printf()

- Format specifiers:
 - `%c` for single characters
 - `%d` for integers
 - `%f` for float/double (fractions): 1234.56
 - `%g` for float/double (scientific): 1.23456E+3
 - `%s` for phrases or 'strings' (coming soon!)

○ Control Characters (Escape Sequences)

Character	Description
<code>\n</code>	newline
<code>\t</code>	horizontal tab
<code>\v</code>	vertical tab
<code>\r</code>	carriage return
<code>\x41</code>	hexadecimal number, 0x41
<code>\101</code>	octal number 101
<code>\0</code>	null character - indicates the end of a string
<code>\'</code>	single quotation mark (')
<code>\"</code>	double quotation mark (")
<code>\\</code>	backslash mark (\)
<code>\b</code>	backspace
<code>\f</code>	formfeed - next page (used for printer)
<code>\a</code>	alert - produce a beep sound

Figure 2-15: Output specification for inventory report

```
printf ("Part Number\tQty On Hand\tQty On Order\t\tPrice\n");
```

Part Number	Qty On Hand	Qty On Order	Price
031235	22	86	\$ 45.62
000321	55	21	\$ 122.00
028764	0	24	\$ 0.75
003232	12	0	\$ 10.91

End of Report

Part numbers must have leading zeros

Leading zeros suppressed

Decimal points must be aligned

Keyboard input: scanf ()

- **scanf ()** will scan formatted input from the keyboard.
- It uses the same format specifiers as printf()
- To read an integer:

```
int num_students;  
scanf ("%d", &num_students);
```

Specifier for
"read an integer value"

VERY IMPORTANT
special symbol

"Place value into this
variable"

format specifiers for `scanf()`

- Format specifiers:
 - **%c** for single characters
 - `scanf("%c", &some_character);`
always put a space between " and % when reading characters
 - **%d** for integers
 - `scanf("%d", &some_integer);`
 - **%f** for float
 - `scanf("%f", &some_float);`
 - **%lf** for double
 - `scanf("%lf", &some_double);`

Variables, literals, constants

- Variable = named memory location that holds a changeable value
- Literal = fixed value written into a program
- Constant = named memory location that holds a non-changeable value

Literals

- Literal = fixed value written into a program
 - Not declared, no memory location assigned
 - Not re-usable; just written directly into each statement
 - This makes it easy to make a mistake
 - Solution: use #define directive.

NOTE : no semicolon after preprocessor directives!

```
#define PI 3.14 ← directs preprocessor to replace all occurrences of PI with 3.14
int main ()
{
    double area, radius;
    radius = 12.1;
    area = PI * radius * radius;
    return 0;
}
```

#define

Syntax:

```
#define NAME value
```

the name is usually capitalized

preprocessor directives are not statements. There must be **no semicolon** at the end.

#define

Advantage : if you need to change the value, you only have to do it once

```
#define PI 3.14
int main ()
{
    double area, radius;
    double circumference;
    radius = 12.6;
    area = PI * radius * radius;
    circumference = 2 * PI * radius;
    return 0;
}
```

if you want to change the value of PI to 3.14159, you only have to do it here and the preprocessor will take care of the rest.

Constants

Constant = named memory location that holds a **non-changeable** value

- Must be declared
- Re-usable
- **MUST** be initialized
- Can not be modified after initialization

```
int main ()
{
    const double pi = 3.14;
    double area, radius;
    radius = 12;
    area = pi * radius * radius;
    pi = 3.14159; /* but, this is wrong */
    return 0;
}
```