ESc101: Variable Types

Instructor: Krithika Venkataramani
Semester 2, 2011-2012

The content of these slides are taken from the lecture slides of Prof. Arnab Bhattacharya, Prof. R.K. Ghosh, Prof. Dheeraj Sanghi and Prof. Manindra Agrawal

Variables

- Variables signify data that may be modified
- Name of a variable can contain letters, digits and underscore _
  - Example: i, y2k, big_name, bigger_name_2
- Case-sensitive: camel, CAMEL and CaMeL are different
- Name cannot start with a digit
  - Example: 1d is not valid
- Name can start with an underscore, but do not do so
  - Example: avoid valid names such as _bad
- Certain keywords are special
  - They are reserved and cannot be used
  - Example: main, if

Types of Variables

- Each variable ‘type’ represents the domain of values
  - Integer: int or char
  - Character: char
  - Boolean: int or char
  - real: double or float
- However, they can store only a subset of the domain
  - int can store numbers from \(-2^{31}\) to \(2^{31}-1\)
- Initial values of variables are specified as constants of the same type
  - int i = 0;
  - double d = 1.4;
  - char c = 'A'
Example Program to Add two numbers

/* Program to add two numbers */
#include <stdio.h> // Include headers
void main() // Main function
{
    int a=2, b=3, c; // Declare variables
    scanf("%d", &a); // Read 'a' from keyboard
    scanf("%d", &b); // Read 'b' from keyboard
    c = a + b;
    printf("%d\n", c); // Write 'c' to screen
}

Different Data Types

- Different types are needed because one type is not suitable for representing data of another type.
- Mixing types may result in precision loss, overflow, underflow.
- Application performance suffers while performing numerically intensive computation if inappropriate data types are used.
- Exceptions must be handled explicitly or they lead to errors.
- Use of appropriate type is important both for efficiency and correctness.

Integer Types

- Two different int types: signed and unsigned
- Maximum signed int in 16 bit: 011111111111111, i.e., 2^15 - 1
- Maximum unsigned int in 16 bit: 111111111111111, i.e., 2^16 - 1
- Possible types to suit our needs are:
  - short int, unsigned short int, unsigned int, long int, unsigned long int.

Integer Representation

- We represent integers as binary numbers
  - For example: 56 will be stored as 00111000
- How do we store negative numbers?
  - 1st bit of memory is usually the sign bit
  - In case of 8 bit memory, only 7 bits are for magnitude.
  - Similarly 32-bit memory would have 31 bits for magnitude
  - Hence the largest positive integer that can be stored in an integer variable on our PCs is: 2^31 - 1.
- Smallest number:
- There are variations on storing magnitude
- Overflow: Trying to store numbers outside the range
Different Real Number Types

- Real numbers of arbitrary precision cannot be represented
- Different types: float, double, long double
- Double is more accurate than float
  - \( \frac{1}{3} \) is printed as 0.3333334326.. as a float, but 0.3333333333.. as a double
- Double is used for precision critical calculations
- By default floating point constants are stored as a double.
- To force float constant should be suffixed with f, i.e., 7.5f or 7.5F.
- Format specifier "%.lf", "%.Lf" are used for using double and long double using scanf/printf

Representation of Real Numbers

- Use the scientific notation: \( f \times b^k \)
- With this notation, we need to store \( f \) and \( k \).
- We also need to decide the value of 'b'.
  - Use 1 bit for sign
  - Value of b is taken as 2
  - Use 8 bits to store k (called exponent)
  - Use 23 bits to store f (called mantissa), in normalized form with integer part of the fraction to be exactly 1 (e.g. 1.0011)
- Exponent can be from -127 to +126
- So the range is from \( 2^{-127} \) to \( 2^{126} \), or \( 10^{-38} \) to \( 10^{38} \) approx.

Errors in representing real numbers

- There are three types of errors:
  - Underflow: Trying to store exponent less than -127
  - Overflow: Trying to store exponent more than 126
  - Rounding off: Storing the nearest floating point number
- Floating point arithmetic
  - The hardware has to do a lot more for floating point arithmetic compared to integer arithmetic
- Do not store numbers as floating point, unless you really need fractions

Range of different data types

- Variables are stored in a predefined space
- A unit of storage is a Byte
- A Byte has space to store a sequence of 8 binary digits
- Different variable types have different storage space assigned
- Assignment of space is machine dependent

<table>
<thead>
<tr>
<th>Type</th>
<th>Space assigned in Bytes</th>
<th>Range</th>
</tr>
</thead>
<tbody>
<tr>
<td>char</td>
<td>1</td>
<td>-2^1 to (2^1-1)</td>
</tr>
<tr>
<td>unsigned char</td>
<td>1</td>
<td>0 to (2^1-1)</td>
</tr>
<tr>
<td>short int</td>
<td>2</td>
<td>-2^1 to (2^1-1)</td>
</tr>
<tr>
<td>unsigned short int</td>
<td>2</td>
<td>0 to (2^1-1)</td>
</tr>
<tr>
<td>int</td>
<td>4</td>
<td>-2^3 to (2^3-1)</td>
</tr>
<tr>
<td>unsigned int</td>
<td>4</td>
<td>0 to (2^3-1)</td>
</tr>
<tr>
<td>float</td>
<td>4</td>
<td>(approx) ±[10^-38, 10^38]</td>
</tr>
<tr>
<td>double</td>
<td>8</td>
<td>(approx) ±[10^-38, 10^38]</td>
</tr>
</tbody>
</table>
Input and output of variables

Correct type specification must be used

<table>
<thead>
<tr>
<th>Type</th>
<th>Format Specifier</th>
</tr>
</thead>
<tbody>
<tr>
<td>char</td>
<td>%c</td>
</tr>
<tr>
<td>int</td>
<td>%d</td>
</tr>
<tr>
<td>unsigned int</td>
<td>%u</td>
</tr>
<tr>
<td>float</td>
<td>%f, %g, %e</td>
</tr>
<tr>
<td>double</td>
<td>%lf</td>
</tr>
</tbody>
</table>

- scanf is for input
  - Format: scanf("< specification >", &<name>);  
    - E.g. c is a char: scanf("%c", &c);

- printf is for output
  - Format: printf("< specification >", <name>);  
    - E.g. c is a char: printf("%c", c);

Character type

- Variable type 'char' used for representing characters
- Characters are special integers of much shorter size
  - Only 256 characters can be represented
- Digits 0-9 are not represented by 00000000 - 00001000
- 0-9 represented by a continuous sequence
- Similarly A-Z (a-z) also represented by a continuous sequence
- ASCII character set is most widely used
  - specifies a standard that maps characters to numbers 0-127
  - Extended ASCII assigns symbols to numbers 128-255
  - ASCII and Extended ASCII use 1 Byte for storage
- Unicode includes characters from all languages of the world
  - Unicode uses 2 Bytes

The ASCII Table

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0000</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0001</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>A</td>
<td>1010</td>
<td>10</td>
<td>A</td>
</tr>
<tr>
<td>Z</td>
<td>1011</td>
<td>11</td>
<td>B</td>
</tr>
</tbody>
</table>

Printing the Code of a Character

/*Program to print the code of a character*/
#include <stdio.h>
void main()
{
    int code; //Declare variable to store the code
    code = (int) getchar(); //Asking user to input the character
    printf("%d", code); //printing the code of the character
}
### Additional data types

<table>
<thead>
<tr>
<th>Data type</th>
<th>Format specifier</th>
<th>Size (machine dependent)</th>
<th>Range</th>
</tr>
</thead>
<tbody>
<tr>
<td>int</td>
<td>%d(decimal), %i</td>
<td>4 bytes</td>
<td>-2^31 -1 to 2^31 -1</td>
</tr>
<tr>
<td>unsigned int</td>
<td>%u</td>
<td>4 bytes</td>
<td>0 to 2^32 -1</td>
</tr>
<tr>
<td>short int (unsigned)</td>
<td>%hd (%hu)</td>
<td>2 bytes</td>
<td>-2^15 -1 to 2^15 -1</td>
</tr>
<tr>
<td>long int (unsigned)</td>
<td>%ld (%lu)</td>
<td>8 bytes</td>
<td>-2^63 -1 to 2^63 -1</td>
</tr>
<tr>
<td>char</td>
<td>%c</td>
<td>1 byte</td>
<td>-128 to 127</td>
</tr>
<tr>
<td>unsigned char</td>
<td>%u, %d</td>
<td>1 byte</td>
<td>0 to 255</td>
</tr>
<tr>
<td>string</td>
<td>%s</td>
<td>array of characters</td>
<td>--</td>
</tr>
<tr>
<td>float</td>
<td>%f, %g, %e</td>
<td>4 bytes</td>
<td>3.4x10^-38 to 3.4x10^38</td>
</tr>
<tr>
<td>double</td>
<td>%lf, %lg, %le</td>
<td>8 bytes</td>
<td>1.7x10^-308 to 1.7x10^308</td>
</tr>
<tr>
<td>long double</td>
<td>%Lf, %Lg, %Le</td>
<td>16 bytes</td>
<td>?</td>
</tr>
</tbody>
</table>

1 bit: 1 or 0 1 Byte: 8 bits  Note: for 32-bit machines long int and int are same

### Additional formats for octal and hexadecimal

<table>
<thead>
<tr>
<th>Data type</th>
<th>Format specifier</th>
<th>Display/ Read</th>
</tr>
</thead>
<tbody>
<tr>
<td>unsigned int</td>
<td>%o</td>
<td>unsigned octal integer</td>
</tr>
<tr>
<td>unsigned int</td>
<td>%x, %X</td>
<td>unsigned hexadecimal integer</td>
</tr>
<tr>
<td>unsigned long int</td>
<td>%lo</td>
<td>unsigned octal integer</td>
</tr>
<tr>
<td>unsigned long int</td>
<td>%lx, %lx</td>
<td>unsigned hexadecimal integer</td>
</tr>
<tr>
<td>unsigned short int</td>
<td>%ho</td>
<td>unsigned octal integer</td>
</tr>
<tr>
<td>unsigned short int</td>
<td>%hx, %hx</td>
<td>unsigned hexadecimal integer</td>
</tr>
</tbody>
</table>