Variables, Primitive Types, Operators, and Expressions

September 20\textsuperscript{th} 2005
Outline

• Relate to readings
• Define “Expressions”
• Discuss how to represent “data” in a program
  - variable name
  - variable type
• List and discuss the properties of primitives types
• List and discuss basic operations on variables
Readings

• PCP Chapter 4 (today)
  - Basic Declarations and Expressions

• TCPL Chapter 2 (next session)
  - Types, Operators, and Expressions

• TCPL Section 1.2 (next session)
  - Variables and Arithmetic Expressions
Expressions

- A C program is a sequence of statements
- A C program is a collection of functions and the data those functions operate on
- The building blocks of statements are expressions: combinations of language key words, function calls, operators, and operands that evaluate to a value
Review: What is a Program?

- A program is a sequence of instructions that operates on **data**
- A C program is a collection of **variables** and **functions** that process the data held in those variables
- Computers process long strings of 1's and 0's
  - need a way to refer to portions of those strings as higher-level data objects
Variables
A variable is a container for data

Unstructured Data

Structured Data

Michael

Patient SSN: 999550000

3.141592658
What exactly is a Variable?

• A variable is the concept of a piece of structured data that can be accessed (read or modified) via well-known, standard rules

• A variable is NOT JUST the data it contains!

• A variable also has:
  - a name or identifier that provides a way to refer to it
  - a type that defines its size (how much memory it uses)
  - a location or memory address specifying where the data is stored
Example:: Simple Integer Values

- Suppose we want to write a program for processing a student's grades
- Need a variable to hold the total score

Variable name: value

total_score: 345
Example:: Variable Declaration

• Declaring a variable is a standard action to let the rest of the program know about a piece of data that will be used.

• The following **program statement** declares (that is, tells the computer to set aside a memory location) an integer variable called 'total_score'

```plaintext
int total_score;
```
Declaring Variables

• Variables are usually declared at the beginning of the program or function they are used in.
• Variable names can be any combination of letters, numbers, or underscores, but must start with a letter or underscore.
• Variable names should be descriptive; avoid names like 'ab', 'x', 'tmp', etc.
• Make sure you don't try to name a variable after a reserved word (if, for, while, switch ...)
A Simple Statement: Initialization

• You can initialize a variable when you declare it
  − This gives the variable a definite “starting value”
  − To initialize a variable, use an assignment statement.

• An assignment statement computes the value of the expression on the right hand side and stores it in the variable named on the left hand side.
  − “=” should be read “stores the result of”

    int total_score = 0;
Initializing a Variable

```c
int total_score = 0;
//...
//...
int main()
{
  //...
}
```

**Computer Memory**

<table>
<thead>
<tr>
<th>address</th>
<th>value</th>
</tr>
</thead>
<tbody>
<tr>
<td>1022</td>
<td></td>
</tr>
<tr>
<td>1023</td>
<td></td>
</tr>
<tr>
<td>1024</td>
<td></td>
</tr>
<tr>
<td>1025</td>
<td></td>
</tr>
<tr>
<td>1026</td>
<td></td>
</tr>
<tr>
<td>1027</td>
<td>00000000</td>
</tr>
</tbody>
</table>

**total_score:**

- Address 1027
- Value 00000000
Subtle Point About Variable Names

• When you program, you see the variable name
• When the computer executes your program, it actually sees the variable memory address
• In both cases, the data is used behind the scenes
Types
Variable Types

• A **type** is a hint to the computer on how to handle the data contained in or referred to by the variable
  - usually this involves size of the storage allocated
• There are 4 primitive (basic) types in C:
  - **int** (regular integer)
  - **char** (1 character)
  - **float** (single precision floating point number)
  - **double** (double precision floating point number)
Type Modifiers

- Types can be augmented by additional information
- Some simple “type qualifiers” are listed below:
  - short (applied to int)
  - long (applied to int)
  - unsigned (only positive values and 0)
  - signed
  - const (specifies that the value can't be changed)
- We usually drop the 'int' when specifying short or long
showsize Demo
Output (sizes are in # of bytes)

[michael@xoren code]$ make showsize
gcc -Wall -g -o showsize showsize.c
[michael@xoren code]$ ./showsize
sizeof(int)         =  4
sizeof(char)        =  1
sizeof(long)        =  4
sizeof(short)       =  2
sizeof(float)       =  4
sizeof(double)      =  8
sizeof(long double) = 12
[michael@xoren code]$
const test Demo
Language Operators
Operators Overview

- You are familiar with many operators from basic math and logic:
  - addition (+), subtraction (-), multiplication (*), division (/)
  - AND (&&), OR (||), NOT (!)

- Operators are basically common functions that take their input and produce some output
- Common enough to have their own symbols in a programming language (see above)
Operators (cont.)

- C has many operators
  - Some you are familiar with (see previous page)
  - Some not: mod, bitwise AND, OR, XOR, relational

- Operators are:
  - unary (take one argument, e.g., !-)
  - binary (take two arguments, e.g., +-*/<>=)
  - ternary (take three arguments)

- Classifications:
  - arithmetic, logic, relational, assignment
Operator Context

• Warning! Operators are represented by symbols. Sometimes, the symbols may mean something completely different based on context. For example:

```java
int x = -1; // the '-' operator is negation

int x = 4 - 3; // the '-' operator is subtraction
```
Arithmetic Operators

- Addition is represented by +
  e.g., sum = x + y;
- Subtraction is rep. by -
  e.g., diff = x - y;
- Multiplication is rep. by *
  e.g., scale = x * y;
- Division is rep. by /
  e.g., quotient = x / y;
- Modulus is rep. by %
  e.g., remainder = x % y;
Relational Operators

- Assignment operator is `=`, e.g., `int sum = x;`
- Equality operator is `==`, e.g., `is_equal = (x==y);`
- Not equal is expressed by `!=`, `not = (x!=y);`
- Less than: `<`
- Greater than: `>`
- Less than or equal to `<=`
- Greater than or equal to `>=`
Logical Operators

- AND: (x&&y)
- OR: (x||y)
- NOT: (!x)
**Bitwise Operators**

- Like logical operators, but operate on the individual bits of a variable, not the whole logical value

```c
int x = 2;
int y = 1;
int r = x || y;
printf("r is: %d",r);
```

```
int x = 2;
int y = 1;
int r = x | y;
printf("r is: %d",r);
```

Output is: r is: 1

Output is: r is: 3
Bitwise Operators

- bitwise OR: |  
- bitwise AND: &  
- bitwise XOR: ^  
- one's complement: ~  
- left shift: <<  
- right shift: >>
Order of Operations

- PEMDAS (power, exponent, mul, div, add, sub)
- For everything else, use parenthesis () to say what you mean
- There are other rules. Learn them at your leisure while using the above two. See table 2.1 in TCPL (page 53)
Type Conversions (TCPL, 2.7)

• Read it.
• Key question is of the form: when I \{add, sub, mul, div, mod\ldots\} an \{int, float, long, \ldots\} \{with, from, by,\ldots\} a \{float, double, long, int\ldots\} what happens?
• Intermediate results are converted according to a set of rules. Basic rule is that results are automatically “graduated” to the type of the larger operand.
Casting

• “casting” is the process of forcing a type conversion
• Below, the integer value in sum is changed into a double type before being used, as is the result of the average score calculation.

```java
int n = 100;
int sum = getsum();
double d = ((double)sum);
double average = (double)sum/n;
```
Things We Haven't Covered Here

- Increment and decrement operators
- Assignment operators
- The ternary condition operator
- Short circuit boolean evaluation
- The nuances of type conversion
- Collections of data types and variables (arrays, next session)