Computer Science 5C
Chapter 6--The Fundamental Data Types

Dr. Scott A. Brandt
Professor, Computer Science Department
Director, UCSC/LANL Institute for Scalable Scientific Data Management

CMPS 5C - 1/7/09
Declarations and Expressions

• Variables and constants are the data that a program manipulates

• Variables and constants have a type
  • Specifies the data storage
  • Specifies the allowed operations

• Expressions are meaningful combinations of constants, variables, operators, and function calls
  • Expressions generally also have a type

• As we have seen, type can affect the value of expressions
# Fundamental Data Types

<table>
<thead>
<tr>
<th>Integral Types</th>
<th>char, signed char, unsigned char short, unsigned short int, unsigned, long, unsigned long</th>
</tr>
</thead>
<tbody>
<tr>
<td>Floating Types</td>
<td>float, double, long double</td>
</tr>
<tr>
<td>Arithmetic Types</td>
<td><em>Integral types and floating types</em></td>
</tr>
</tbody>
</table>
Characters and the data type char

- Chars are just small integral types
  - Usually 1 byte / 8 bits of memory (10011010)
- They can be used to store values that represent characters
- char c = ‘a’; /* ‘a’ has the encoding 97 */
- int i = 65; /* 65 is the encoding of ‘A’ */
- printf(“%c”, c); /* a is printed */
- printf(“%c”, c+1); /* b is printed */
The data type int

• The basic integral type
  • 2 or 4 bytes / 16 or 32 bits, depending upon the machine

• Used to store integers
  • 0, 1, 2, etc.

• 16 bits: -32768 to +32767
• 32 bits: -2147483648 to +2147483647
Integral types short, long, and unsigned

• Like int, but with less or more storage

• Short
  • Usually 2 bytes / 16 bits
  • Takes up less memory

• Long
  • Usually 4 or 8 bytes / 32 or 64 bits
  • Can store larger values

• Unsigned
  • Positive integers only
    • 0 to +65535 or +4294967295
### The floating types

<table>
<thead>
<tr>
<th>Suffix</th>
<th>Type</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>f or F</td>
<td>float</td>
<td>3.7F</td>
</tr>
<tr>
<td></td>
<td>double</td>
<td>3.7</td>
</tr>
<tr>
<td>l or L</td>
<td>long double</td>
<td>3.7L</td>
</tr>
</tbody>
</table>
Examples

• Yes
  • 3.14144
  • 2.7
  • 10.5e7

• No
  • 3,000,000.7 /* Commas aren’t allowed */
  • 5 /* This is an int */
Storage for floating point types

• Float
  • Typically 4 bytes
  • Can store about 6 decimal digits

• Double
  • Typically 8 bytes
  • Can store about 15 decimal digits

• Long Double
  • 8 or 16 bytes
  • 15 or 31 decimal digits
#limits.h

- `#define SCHAR_MAX 127` /* min value for a signed char */
- `#define SCHAR_MIN (-128)` /* max value for a signed char */
- `#define UCHAR_MAX 255` /* max value for an unsigned char */
- `#define CHAR_MAX 127` /* max value for a char */
- `#define CHAR_MIN (-128)` /* min value for a char */
- `#define USHRT_MAX 65535` /* max value for an unsigned short */
- `#define SHRT_MAX 32767` /* max value for a short */
- `#define SHRT_MIN (-32768)` /* min value for a short */
- `#define UINT_MAX 0xffffffff` /* max value for an unsigned int */
- `#define INT_MAX 2147483647` /* max value for an int */
- `#define INT_MIN (-2147483647-1)` /* min value for an int */
• ifdef __LP64__
• #define ULONG_MAX 0xffffffffffffffffUL /* max unsigned long */
• #define LONG_MAX 0x7fffffffffffffffL /* max signed long */
• #define LONG_MIN (-0x7fffffffffffffffL-1) /* min signed long */
• else /* !_LP64__ */
• #define ULONG_MAX 0xffffffffUL /* max unsigned long */
• #define LONG_MAX 2147483647L /* max signed long */
• #define LONG_MIN (-2147483647L-1) /* min signed long */
• endif /* __LP64__ */
• #define ULLONG_MAX 0xffffffffffffffffULL /* max unsigned long long */
• #define LLONG_MAX 0x7fffffffffffffffLL /* max signed long long */
• #define LLONG_MIN (-0x7fffffffffffffffLL-1) /* min signed long long */
The sizeof() operator

- `sizeof(object)` returns the number of bytes of storage used for that object
- `printf("char: %d\n", sizeof(char));`
- `printf("short: %d\n", sizeof(short));`
- `printf("int: %d\n", sizeof(int));`
- `printf("long: %d\n", sizeof(long));`
Mathematical functions

- math.h
- sqrt()
- pow()
- exp()
- log()
- sin()
- cos()
- tan()
#include <stdio.h>
#include <math.h>

void do_work(void);

int main(void)
{
    printf("The square root of x and x raised ");
    printf("into the x power will be computed.");
    printf("---\n\n");

    do_work();

    return 0;
}
void do_work()
{
    double x;

    while(1) {
        printf("Input x: ");
        scanf("%lf", &x);

        if(x >= 0.0) {
            printf("x = %22.15e\n", x);
            printf("sqrt(x) = %22.15e\n", sqrt(x));
            printf("pow(x,x) = %22.15e\n", pow(x,x));
        } else {
            printf("\nSorry, the number must be nonnegative\n\n");
        }
    }
}
Conversions and casts

• Mathematical expressions may operate on objects of different types

• When that happens, the smaller type is converted to the larger type

• Example:
  • int a; double d;
  • a + d has type double
  • a gets promoted to double, then the addition takes place
Usual Arithmetic Conversions

1. If either operand is of type long double, the other is converted to long double
2. Otherwise, if either is of type double, the other is converted to double
3. Otherwise, if either is float -> float
4. Otherwise, if either is unsigned long -> long or unsigned long
5. Otherwise, if either is long -> long
6. Otherwise, if either is unsigned -> unsigned
7. Otherwise both ints
Examples

• char c; short s; int i; unsigned u; unsigned long ul; float f; double d; long double ld;

• \(c - s / i\)
• \(u * 2.0 - i\)
• \(c + 3\)
• \(c + 5.0\)
• \(d + s\)
• \(2 * i / l\)
Casts

• (double) i
• (long) (‘A’ + 1.0)
• x = (float) ((int) y + 1)
• (double)(x = 77)
• (type) has highest precedence
#include <stdio.h>

double compute(double p, double r, int n);
void prn_instructions(void);
void prn_results(double a, double p, double r, int n);

int main(void)
{
    double amount, principal, rate;
    int nyears;

    prn_instructions();
    while (1) {
        printf("Input three items:  ");
        scanf("%lf%lf%d", &principal, &rate, &nyears);
        amount = compute(principal, rate, nyears);
        prn_results(amount, principal, rate, nyears);
    }
}
void prn_instructions(void)
{
    printf("This program computes interest compounded yearly.\n");
    printf("Input principal, interest, and number of years.\n");
    printf("For $1000 at 5.5% for 17 years, for example, you would enter: 1000 5.5 17\n\n");
}

double compute(double principal, double rate, int nyears)
{
    int i;
    double amount = principal;
    rate *= 0.01;
    for(i = 0; i < nyears; i++)
    {
        amount += amount * rate;
    }
    return amount;
}
void prn_results(double a, double p, double r, int n)
{
    double interest = a - p;
    printf("Interest rate: %f\n", r);
    printf("  Time period: %d years\n\n", n);
    printf("Beginning principal: %9.2f\n", p);
    printf("     Interest accrued: %9.2f\n", interest);
    printf("           Total amount: %9.2f\n", a);
}