Primitive Data Types and Terminal I/O

CS2023 Winter 2004

Outcomes: Data types and terminal I/O

- “C for Java Programmers”, Chapter 3
- After the conclusion of this section you should be able to
  - Give the relative size of the primitive data types
  - Find the range of values that the primitive data types can represent
  - Use assignment conversion and arithmetic conversion between data types
  - Modify these conversions using a `cast`
  - Write a basic C program that reads and writes to and from the terminal, using the appropriate idioms for
    - Character I/O with `getchar`, `putchar`
    - Formatted I/O with `scanf`, `printf`

Basic Data Types

C provides several primitive data types: **`char`**, **`int`**, **`float`** and **`double`**.

- No built-in Boolean type; instead use **`int`**:
  the value `0` stands for false,
  any non-zero value stands for true
- No guarantee that a specific amount of memory will be allocated to a particular data type.

Range of Integers

- **Signed integers** use the leftmost bit (sign bit), to represent the sign.
  The largest unsigned 16-bit integer: $2^{16} - 1$
  The largest signed 16-bit integer: $2^{15} - 1$

- C provides a header file **`limits.h`**, it defines e.g.

  `CHAR_BIT` - the width of `char` type in bits (>= 8)

  `INT_MAX` is the maximum value of `int` (>= 32,767).

Integer Types

- plain, signed and unsigned

```
short unsigned int
signed long
int
```

- size(**short**) <= size(**int**) <= size(**long**)

Character Types

- There are three character data types in C:
  (plain) **`char`**
  `unsigned char`
  `signed char`
- Character types are actually represented internally as integers (unsigned or signed).
- Two popular character sets:

  ASCII (American Standard Code for Information Interchange): 7 bits (127 characters)

  EBCDIC (used by IBM computers)

Floating-point Types

`float`
`double`
`long double`

Use **`float.h`** for sizes and ranges.

Only guarantee:

size(**`float`**) <= size(**`double`**) <=
size(**`long double`**)

Declarations of Variables and Constants

```
int i; /* initial value undefined */
double d = 1.23;

const double PI = 3.1415926;
```

sizeof

`sizeof`(type name)

or

`sizeof` expression

returns the size of the data type or object represented by the expression.

```
sizeof(int)
sizeof i
```

Type Conversions

- Type T is **wider** than type S (and S is **narrower** than T), if

  `sizeof(T) >= sizeof(S)`

- The narrower type is *promoted* to the wider type, the wider type is *demoted* to the narrower type
- An **`int`** value can be safely promoted to **`double`**
- A **`double`** value can *not* be safely demoted to **`int`**

Arithmetic Conversions

- If operands of an expression are of different types, then these operands will have their types changed, using *arithmetic conversions*.
- A lower precision type is *promoted* to a higher precision type according to the following hierarchy:

```
int
unsigned
long
unsigned long
float
double
long double
```

Assignment and Cast Conversions

- Assignment conversions occur when the expression on the right hand side of the assignment has to be converted to the type of the left-hand side.
- The **type cast** expression

  (*typ*) exp

  converts the expression **exp** to the type ***typ***.

```
double f, c;
f = 10;      /* assignment conversion */
f = 100.2;
c = (5/9)*(f - 32);
c = ( (double)5/9 ) * (f - 32); /* cast */
```

Type Synonyms: typdef

```
typedef existingType NewType;
```

For example, if you want to use a Boolean type, define

```
typedef int Boolean;
```

```
Boolean b = 1;
```

When specifying a new type using `typedef`, start identifier names with an upper case letter.

Literal Constants

- integer: `123  47857587L`
- floating point: `12.66  23478.78899E-20`
- character: `'\n' 'd'`
- string: `"abc"`

Expressions

- As in Java, but evaluation rules more relaxed. Only four binary operators that guarantee that left operand is evaluated before the right operand:

logical AND, as in	`e1 && e2`
logical OR, as in	`e1 || e2`
a conditional expression, as in	`e1 ? e2 : e3`
a comma expression, as in	`e1, e2`

Terminal I/O

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

`int getchar()` to input a single character

`int putchar(int)` to output a single character

```
/*
 * Program that reads two characters and
 * prints them in reverse order, separated by
 * a tab and ending with end of line.
 * Error checking: Program terminates if
 * either of the input operations fails.
 *  No error checking for the output
 */
#include <stdio.h>
int main()
{
   int c, d;

   if((c = getchar()) == EOF)
      return 1;
```

Idiom!

```
    if((d = getchar()) == EOF)
        return 1;

    putchar(d);
    putchar('\t');
    putchar(c);
    putchar('\n');

    return 0;
}
```

Errors

- Placement of brackets:

```
if(c = getchar() == EOF)
```

- The compiler interprets it as follows:

```
if(c = getchar() == EOF)
```

- **`char c;`**

```
c = getchar()
```

Formatted Output

```
int printf("format", exp)

 printf("%d", 123);

 printf("The value of i = %d.\n", i);

 printf("Hello world\n");
```

Integer Conversions

To print integers, the following conversions are used:

`d`	signed decimal
`ld`	long decimal
`u`	unsigned decimal
`o`	unsigned octal
`x, X`	unsigned hexadecimal

```
printf("%d%o%x", 17, 18, 19);
```

Float Conversions

To print floating point values, the following conversions are used (the default precision is 6):

`f`	`[-] ddd.ddd`
`e`	`[-] d.ddddde{sign}dd`
`E`	`[-] d.dddddE{sign}dd`
`g`	shorter of `f` and `e`
`G`	shorter of `f` and `E`

```
printf("%5.3f\n", 123.3456789);
printf("%5.3e\n", 123.3456789);
123.346
1.233e+02
```

String Conversions

To print characters and strings, the following conversions are used:

- `c` character
- `s` character string

```
printf("%c", 'a');
printf("%d", 'a');
printf("This %s test", "is");
```

Formatted Input

```
int scanf("format", &var)
```

```
int i;
double d;

scanf("%d", &i);
scanf("%lf", &d);

scanf("%d%lf", &i, &d);
```

More on scanf

`scanf()` returns the number of items that have been successfully read, and `EOF` if no items have been read and end-file has been encountered.

For example **`scanf("%d%d", &i, &j)`**

returns the following value:

2	if both input operations were successful
1	if only the first input operations was successful
0	if the input operation failed
`EOF`	if an end-of-file has been encountered.

```
int main() {
    double i, j;

    printf("Enter two double values:");
    if(scanf("%lf%lf", &i, &j) != 2)
        return 1;

    printf("sum = %f\ndifference = %f\n",
                  i + j, i - j);

    return 0;
}
```

← Idiom!

Idioms

- Read single character:

```
if((c = getchar()) == EOF) …
    /* error, else OK */
```

- Read single integer with prompt:

```
printf("Enter integer: ");
if(scanf("%d", &i) != 1 ) …
/* error, else OK */
```

- Read two integers:

```
printf("Enter integer: ");
if(scanf("%d", &i) != 1 ) …
/* error, else OK */
```