CS1100 Introduction to Programming

Arrays

Madhu Mutyam
Department of Computer Science and Engineering
Indian Institute of Technology Madras

An Array

- A data structure containing items of same data type
- Declaration: array name, storage reservation
  - *int* marks[7] = {22,15,75,56,10,33,45};
    - a contiguous group of memory locations
    - named “marks” for holding 7 integer items
  - elements/components - variables
    - marks[0], marks[1], … , marks[6]
    - marks[*i*], where *i* is a position/subscript ($0 \le i \le 6$)
  - the value of marks[2] is 75
  - new values can be assigned to elements
    - marks[3] = 36;

22	0
15	1
75	2
56	3
10	4
33	5
45	6

Example using Arrays

Read ten numbers *into an array* and compute their average

```
#include <stdio.h>
int main( ){
   int numbers[10], sum = 0, i;
   float average;
   for ( i = 0; i < 10; i++)
        scanf(“%d”, &numbers[i]);
   for ( i = 0; i < 10; i++)
        sum = sum + numbers[i];
   average = (float) sum/10;
   printf(“The average of numbers is: %f”, average);
   return 0;  /* should be there in all programs */
}
```

Counting Digits in Text (Kenighan & Ritchie, pp. 59)

An array of ten integers

```
#include<stdio.h>
int main( ){
  int c, i, nWhite, nOther, nDigit[10];
  nWhite = nOther = 0;
  for(i=0;i<10;i++)
       nDigit[i]=0;
  while((c = getchar( ))! = EOF){
       switch(c){
          case ‘0’:case ‘1’:case ‘2’:case ‘3’:case ‘4’:case ‘5’:
          case ‘6’:case ‘7’:case ‘8’:case ‘9’: nDigit[c- ‘0’]++;
                                          break;
```

… Counting digits

```
          case ‘  ’:
          case ‘\n’:
          case ‘\t’: nWhite++; break;
          default : nOther++; break;
       }
   }
   printf(“Digits: \n”)
   for(i=0;i<10;i++)
        printf(“%d occurred %d times \n”, i, nDigit[i]);
   printf(“White spaces: %d, other: %d\n”, nWhite,
               nOther);
}
```

Polynomial Evaluation

Evaluate
$p(x) = a_n x^n + a_{n-1}x^{n-1} + a_{n-2}x^{n-2} + \ldots a_1 x + a_0$ at a given x value.

Computing each term and summing up
$n + (n-1) + (n-2) + \ldots + 1 + 0 = n(n+1)/2$ multiplications and n additions

Improved Method:
$p(x) = a_0 + x(a_1 + x(a_2 + x(a_3 + \ldots + x(a_{n-2} + x(a_{n-1} + xa_n))\ldots)))$
for instance, $p(x) = 10x^3 + 4x^2 + 5x + 2$
$= 2 + x(5 + x(4 + 10x))$
n multiplications and n additions – will run faster!

… Polynomial Evaluation

```
#include <stdio.h>
main( ){
 int coeff[20], n, x, value, i;         /*max. no. of coeff ’s is 20*/
 scanf(“%d%d”, &n, &x);   /*read degree and evaluation point*/
 for(i = 0; i <= n; i++)
     scanf(“%d”, &coeff[i]);            /* read in the coefficients */
                                        /* a0, a1, … , an */
 value = coeff[n];                      /* an */
 for(i = (n-1); i >= 0; i--)            /* evaluate p(x) */
     value = x*value + coeff[i];
 printf(“The value of p(x) at x = %d is %d\n”, x, value);
}
```

More Exercises

- Sort an array of numbers into ascending order.
  - Write the output into another array
  - Assuming that arrays are expensive, use only one array: read in the values into an array, sort in place, and print out the array.
- Matrix Sorting – The input is a matrix. Identify a sequence of column interchanges such that in the resulting matrix the rows are all sorted in ascending order. Can every matrix be sorted?

Data, Types, Sizes, Values

- *int, char, float, double*
- *char* – one byte, capable of holding one character
- *int* – an integer, different kinds exist!
  - Integer Qualifiers – short and long
  - *short int* – 16 bits, *long int* – 32 bits (Typical)
  - Size is compiler dependant
    - based on the underlying hardware
  - *int* is at least 16 bits, *short* is at least 16 bits, *long* is at least 32 bits
  - *int* is no larger than *long* and at least as long as *short*

The Char, Signed and Unsigned Types

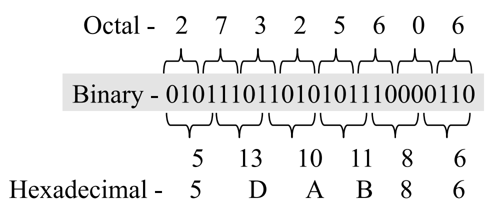
- Qualifier signed or unsigned can be applied to *int* or *char*
- Unsigned numbers are non-negative
- Signed *char* holds numbers between –128 and 127
  - Whether *char* is signed or unsigned depends on the system. Find out on your system.
  - Print integers between 0 to 255 as characters, (and also integers between –128 to 127) on your system.

Number Systems

- Decimal (base 10 – uses 10 symbols {0..9})
  - 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13 …
- Unary (base 1)
  - 0, 00, 000, 0000, 00000 …
- Binary (base 2) – uses 2 symbols {0,1})
  - 0, 1, 10, 11, 100, 101, 110, 111, 1000, 1001, 1010 …
- Octal (base 8 – start with a 0 in C)
  - 0, 1, 2, 3, 4, 5, 6, 7, 10, 11, 12, 13…
  - 00, 01, 02, 03, 04, 05, 06 … C treats them as octal
- Hexadecimal (base 16 – start with 0x)
  - 0, 1, …, 9, A, B, C, D, E, F, 10, 11, … 19, 1A, 1B, …

Binary, Octal and Hexadecimal

- The internal representation of binary, octal, and hexadecimal numbers is similar



A Funny Infinite Loop - Arithmetic

- This program of mine, ran into an infinite loop. I only wanted to find which numbers corresponded to which characters, the significance of signed and unsigned characters, basically relationship between which integers can be printed as characters we recognize. Why did the infinite loop happen, how to avoid it?

```
#include<stdio.h>
main( ){
   char c;
   for(c=-128; c <= 127; c++) printf(“%d -- %c \n”, c, c);
```

Print it as a decimal number

Print it as a character

Try omitting one parameter…

Float and Double

- Two types, one for single-precision arithmetic, and the other for double precision arithmetic
- Long double is used for extended-precision arithmetic
- The size of floating pointing objects are implementation defined

Variable Initialization

- Variables may be initialized either at the time of declaration
  - Example: #define MAXLINE 200
    char esc = ‘\\’;
    int i = 0;
    int limit = MAXLINE + 1;
    float eps = 1.0e-5
- Or they may be assigned values by assignment statements in the program
- Otherwise they contain some random values

Constants

- At run time, each variable holds a value, which changes from time to time
- Constant has a value that does not change
- 1234 is of type int
- 123456789L is a long constant
- 123456789ul is an unsigned long constant
- 123.4 is a floating point constant, so is 1e-2 which denotes .01. Their type is double.
- If suffixed by an f, or by l, the type is float or long double, respectively

Character Constants …

- …are integers, written as one character within single quotes. Example – ‘a’, ‘x’, ‘1’, ‘2’ etc.
- Value of a character constant is the numeric value of the character in the machine’s character set.
  - For example, ‘1’ has the value 49 in the ASCII character set
- Character constants can participate in arithmetic
  - What does ‘1’+ ‘2’ hold? (not ‘3’!)
    - Understand this distinction
  - Character arithmetic is used mainly for comparisons

Characters – escape sequences

\a	alert (bell)	\\	backslash
\b	backspace	\?	question mark
\f	formfeed	\’	single quote
\n	newline	\”	double quote
\r	carriage return	\0oo	octal number
\t	horizontal tab	\xhh	hexadecimal
\v	vertical tab		

Non-printable characters

Constants Expressions

- Expressions all of whose operands are constants
- These can be evaluated at compile time
- Examples:

  #define NUM_ROWS 100

  #define NUM_COLS 100

  #define NUM_ELTS NUM_ROWS*NUM_COLS
- ***#define*** is preprocessor directive (recall ***#include***)

Enumerated Constants

By default enum values begin with 0

- ***enum*** boolean {No, Yes};
  - defines two constants No = 0, and Yes = 1.
- ***enum*** months {jan = 1, feb, march, april, may, jun, jul, aug, sep, oct, nov, dec};
  - when a value is explicitly specified (jan=1) then it starts counting from there
- ***enum*** escapes {BELL = '\a', BACKSPACE = '\b', TAB = '\t', NEWLINE = '\n' };
  - more than one value can be specified

enum and #define

- Better than ***#define***, the constant values are generated for us
  - Values from 0 onwards unless specified
  - Not all values need to be specified
  - If some values are not specified, they are obtained by increments from the last specified value
- Variables of ***enum*** type may be declared
  - but the compilers need not check that what you store is a valid value for enumeration

Declaring Constants

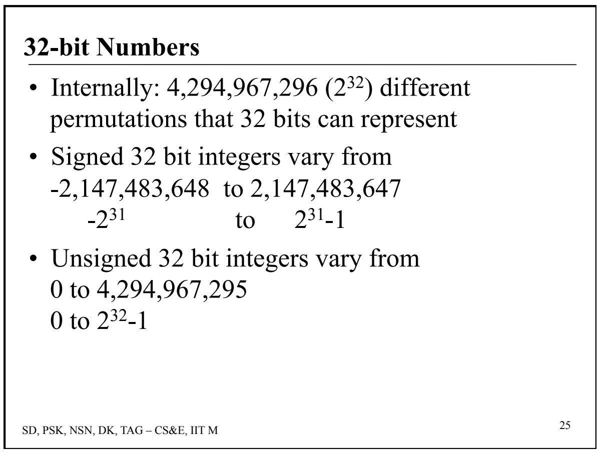
- The qualifier ***const*** applied to a declaration specifies that the value will not be changed.
  - Example: ***const int*** J = 25; /* J is a constant through out the program */
- Response to modifying J depends on the system. Typically, a warning message is issued while compilation.
  - Example: ***const char*** MESG[] = "how are you?";
  - The character array MESG is declared as a constant which will store "how are you?"

Strings

- A string is a array of characters terminated by the null character, '\0'
- A string is written in double quotes
  - Example: "This is a string"
- " " – empty string
- Anything within single quotes gets a number associated with it
- 'This is rejected by the C Compiler'
- Exercise: understand the difference between 'x' and "x"

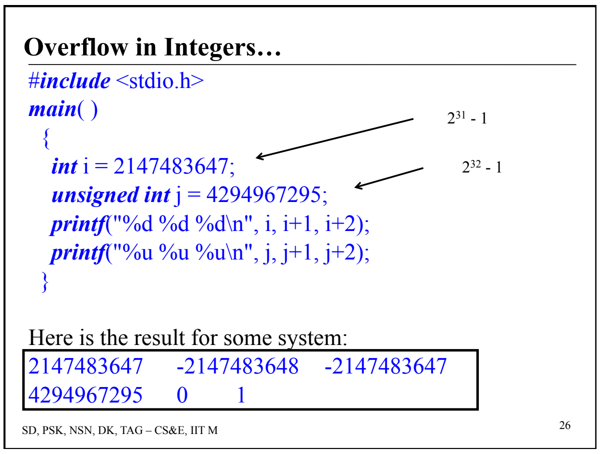
Functions to Handle Strings

- Strings
  - a non basic data type, a constructed data type
  - we require functions to handle them
- Typical functions are:
  - Length of a string; string comparison; string concatenation, etc.
- Standard functions are provided with *string.h*
  - *strlen*( ); *strcmp*( ); *strcpy*( ); *strncpy*( ); *strcat*( ); *strncat*( );



32-bit Numbers

- Internally: 4,294,967,296 (2^{32}) different permutations that 32 bits can represent
- Signed 32 bit integers vary from
  -2,147,483,648 to 2,147,483,647
  -2^{31} to $2^{31}-1$
- Unsigned 32 bit integers vary from
  0 to 4,294,967,295
  0 to $2^{32}-1$



Overflow in Integers…

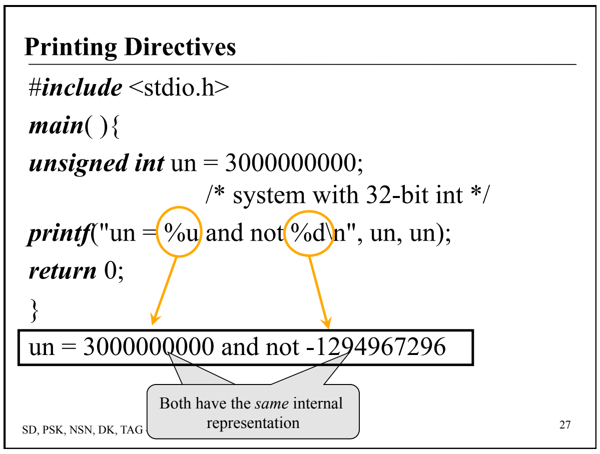
```
#include <stdio.h>
main( )
 {
  int i = 2147483647;
  unsigned int j = 4294967295;
  printf("%d %d %d\n", i, i+1, i+2);
  printf("%u %u %u\n", j, j+1, j+2);
 }
```

2^{31} - 1

2^{32} - 1

Here is the result for some system:

```
2147483647    -2147483648    -2147483647
4294967295    0      1
```

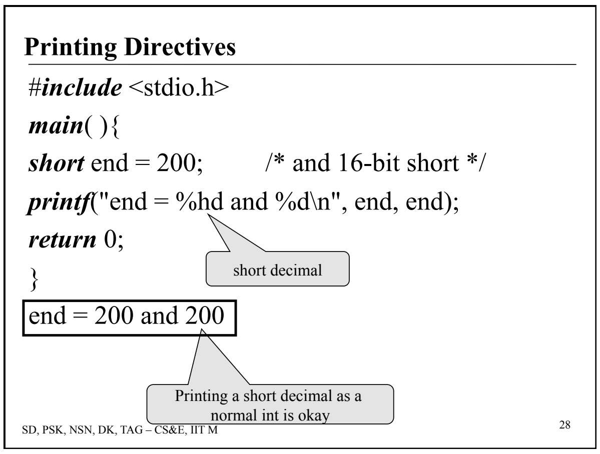


Printing Directives

```
#include <stdio.h>
main( ){
unsigned int un = 3000000000;
                       /* system with 32-bit int */
printf("un = %u and not %d\n", un, un);
return 0;
}
```

```
un = 3000000000 and not -1294967296
```

Both have the *same* internal representation



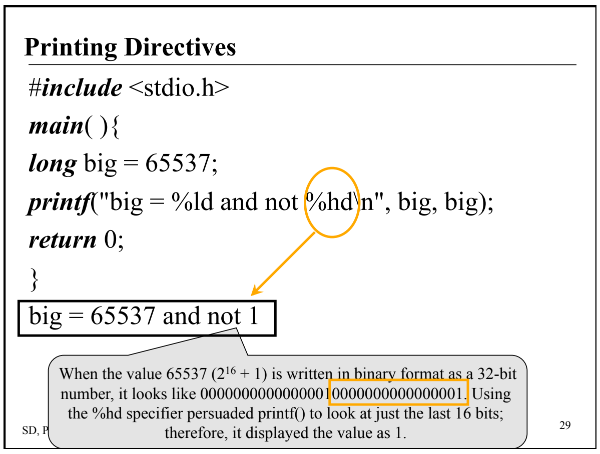
Printing Directives

```
#include <stdio.h>
main( ){
short end = 200;         /* and 16-bit short */
printf("end = %hd and %d\n", end, end);
return 0;
}
```

short decimal

```
end = 200 and 200
```

Printing a short decimal as a normal int is okay

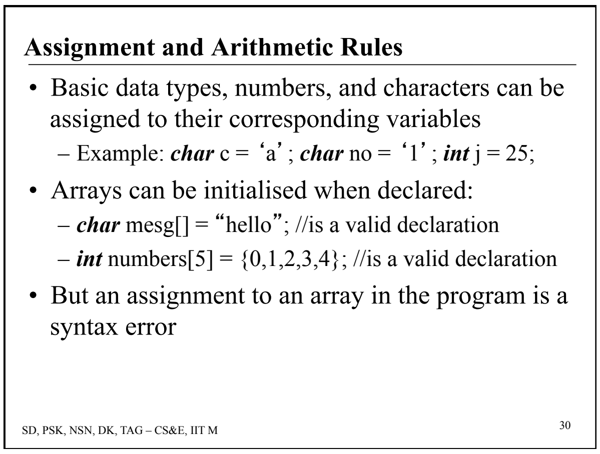


Printing Directives

```
#include <stdio.h>
main( ){
long big = 65537;
printf("big = %ld and not %hd\n", big, big);
return 0;
}
```

```
big = 65537 and not 1
```

When the value 65537 (2^{16} + 1) is written in binary format as a 32-bit number, it looks like 0000000000000001$\boxed{0000000000000001}$. Using the %hd specifier persuaded printf() to look at just the last 16 bits; therefore, it displayed the value as 1.



Assignment and Arithmetic Rules

- Basic data types, numbers, and characters can be assigned to their corresponding variables
  - Example: ***char*** c = ‘a’ ; ***char*** no = ‘1’ ; ***int*** j = 25;
- Arrays can be initialised when declared:
  - ***char*** mesg[] = “hello”; //is a valid declaration
  - ***int*** numbers[5] = {0,1,2,3,4}; //is a valid declaration
- But an assignment to an array in the program is a syntax error

Recap

- Variables
- Assignments
- Relational operators (comparisons)
- Selection and repetition constructs: control structures
- Data types and their limitations
- Arrays : arrayName[n], single dimensional array
  - arrayName[m][n] – 2D array
  - arrayName[i][j] gives the element in the i-th row and j-th column

Logical Operators

- Recall relational operators {<=, <, >, >=} to compare values
- Boolean AND (&&) and Boolean OR (||)
  - Expressions involving these as operations take Boolean values, and their operands also take Boolean values
    - Called truth values also
  - E1&&E2 is true if and only if both E1 and E2 are true
  - E1||E2 is true if and only if either E1 or E2 or both are
- Precedence of && is higher than ||, and both are lower than relational or equality operators

How to use these in a program

- They are used when composite conditions are to be tested in decision statements

```
for(i=0;i < lim–1&&(c=getchar())!= ‘\n’ &&c!=EOF;i++)
        s[i] = c;
```

- The loop is executed as long as all the test conditions are true
- The loop is exited when any test condition becomes false
  - For example when an <Enter> is read from keyboard

Operators

- Increment operator: effect is to increment value of a variable
  - *x* = *j*++ //*x* gets the value of *j*, and then *j* is incremented
  - *x* = ++*j* //*j* is incremented first, then assigned to *x*
- Decrement operators – decrements values
  - *x* = *j*-- //*x* gets the value of *j*, then *j* is decremented by 1
  - *x* = --*j* //*j* is first decremented, and then assigned to *x*
- Assignment operator short cut
  - E_1 op= E_2 is equivalent to the assignment E_1 = E_1op E_2
    - x -= 5; same as: x = x – 5;

Exercise

- Write a program which will exit when a certain number of occurrences of any keystroke is read.
  - You need arrays
  - Loops
  - Loops with logical operations and so on.