Principles of Computer Science II

Nadeem Abdul Hamid

CSC121A - Spring 2005

Lecture Slides 11 - C Functions and Structured Programming

Review

- Relational, equality, and logical expressions evaluate to `int` values `1` (true) or `0` (false)
- Expressions are parsed according to precedence and associativity rules
  - Rules of parsing C are standardized; order of evaluation is not (except for , ? && || operators)
- Statement forms
  - sequence, empty, assignment, compound (block), conditional (if/if-else/switch), looping (for/while/do-while), goto, continue, break

Structured Programming

- A problem-solving strategy and programmming methodology
  - Flow of control be as simple as possible
  - Program construction using top-down design
- Top-down design (stepwise refinement)
  - Repeatedly decompose problem into smaller problems, until you have a collection of small problems or tasks which can be invidually coded very easily
- Code for this decomposition is written using the C *function* mechanism (similar to methods in Java)

Histogram Program

- Write a program that displays a histogram (bar chart of *s) based on input read from a file. The file contains an arbitrarily-long list of numbers between 1 and 30. The last number in the list is followed by a -1.

Histogram Program

```
/*
 * histogram.c
 * Reads input and displays a histogram of stars
 * Input should be in the range 1 to 30
 * Nadeem Abdul Hamid
 */

/* #define WITHLABELS */

#include <stdio.h>

/* function prototype */
void printStars( int );

/* main function */
int main( void ) {
  int n, res;

  res = scanf( "%d", &n );   /* priming read */
  while ( res == 1 && n > 0 && n <= 30 ) {
    printStars( n );
    res = scanf( "%d", &n ); /* read next input value */
  }

  /* error if input conversion problem, or if n is not -1 or in the
     proper range [1 ... 30] */
  if ( res != 1  ||  n != -1 ) {
    printf( "Abnormal program termination: invalid input.\n" );
    return -1;
  }

  return 0;
} /* end main */

/* printStars prints k stars and then a newline */
void printStars( int k ) {
#ifdef WITHLABELS
  printf("%3d: ", k);
#endif
  for ( ; k > 0; --k ) printf("*");
  printf("\n");
}
```


C Functions

- "Called" ("Invoked") using the name of the function followed by parentheses
- Definition:

```
return-type function-name ( parameter-type-list )
{
        declarations
        statements
}
```

Function Prototypes

return-type function-name (*parameter-type-list*) ;

- Used to declare functions before they are used
  - Identifiers in the type list are for documentation only - ignored by compiler
  - Variable number of arguments specified using … (`printf` function)
- One of most important improvements of ANSI C over traditional C
  - Allow compiler to validate function calls
  - Values passed to functions are coerced, as necessary


```
/* try compiling with gcc -Wall …  (turns on all warnings) */

#include <stdio.h>

int funcB( int );
int funcC( float x );

int main(void) {

        printf("%f\n", funcA( 3.7 ));
        printf("%d\n", funcA( 3.7 ));

        printf("%f\n", funcB( 3.7 ));  /* try  (float) cast */
        printf("%d\n", funcB( 3.7 ));

        return 0;
}

int funcA( int n ) { return n * n; }

int funcB( int n ) { return n * n; }

int funcC( int x ) { return x * x; }
```

Parameter type list allows compiler to coerce (convert) arguments to proper data type

Conflicting prototype and definition results in compilation error

By default, C assumes `int` *return type*

Compiler's View of Functions

- Function declarations generated in different ways by the compiler
  - Function call
    - Compiler assumes default declaration, returning `int` and no assumptions about parameters
  - Function definition
    - ANSI C style gives return type and parameter types
  - Function prototype
    - Special case of function declaration
    - Header files mostly contain these prototypes

Declarations, Prototypes, Definitions

- *Function declaration* specifies interface between function and rest of world (return type, argument types)
- *Function prototype* is an ANSI-style function declaration
- *Function definition* gives same info as declaration with names for arguments and block of code

Other Features of C Functions

- If no return type is specified for a function, compiler assumes `int`
  - but it is better style to always indicate the return type
  - Similarly, if a parameter's type is not specified, the compiler assumes `int`

C Standard Library

- Be familiar with the functions in the library (Appendix A)
- Whenever possible, reuse functions from the C library
  - Reduces development time
  - Increases program portability

Standard Library Headers

- <assert.h>
  - Macros and information for diagnostics and debugging
- <ctype.h>
  - Character test functions, conversions (upper-to-lowercase)
- <errno.h>
  - Macros for reporting error conditions
- <float.h>
  - Floating point size limits
- <limits.h>
  - Integral size limits
- <locale.h>
  - Prototypes and info for modifying locale of program - date, time format, etc.
- <math.h>
  - Math library functions
- <time.h>
  - Functions and types for manipulating time and date
- <setjmp.h>
  - Functions allowing bypassing of usual fn. call and return sequence
- <signal.h>
  - Functions and macros handling various program conditions that may arise
- <stdarg.h>
  - Dealing with variable argument functions
- <stddef.h>
  - Common C type definitions
- <stdio.h>
  - Standard I/O functions and related information
- <stdlib.h>
  - Number-text conversion, memory allocation, random numbers, other utilities
- <string.h>
  - String processing functions

Call-by-Value / Call-by-Reference

- Two forms of argument passing common in programming languages
- By value: a *copy* of the argument's value is made and passed to the called function
  - Changes to copy do not affect original value in calling function
- By reference: Caller allows called function to modify the variable's value
  - C simulates call-by-reference by passing addresses and pointers as arguments (arrays are always passed by reference)

Java Method Invocation

- Call-by-value: primitive type arguments
- Call-by-reference: object type arguments

```
class Pair {
    int a;
    int b;
    Pair(int aa, int bb) { a = aa; b = bb; }
}

public class Calls {
    public static void main(String args[]) {
        int i = 5;
        int j = 9;
        Pair q = new Pair( i, j );

        System.out.println( q.a + " ... " + q.b );
        System.out.println(   i + " ... " + j   );
        swapA( q );
        swapB( i, j );
        System.out.println( q.a + " ... " + q.b );
        System.out.println(   i + " ... " + j   );
    }

    static void swapA( Pair p ) {
        int t = p.a;
       p.a = p.b;
       p.b = t;
    }

    static void swapB( int a, int b ) {
        int t = a;
        b = a;
        a = t;
    }
}
```


Programming Style

- Programs should be written as collections of small, well-designed functions
  - In most (large) programs, `main` consists of calls to other functions that perform bulk of the program's work
- Each function should be limited to performing a *single, well-defined* task
  - Function name should express the task clearly
  - If you cannot choose a concise name for the function, it is probably trying to do too much - break it up into smaller functions
- Functions should be no longer than one page
  - Better yet, they should be no longer than half a page (15-20 lines of code)
- Functions requiring large number of parameters may be performing too many tasks
- A value-returning function should have only one (or very few) `return` statement

Random-Number Generation

- `<stdlib.h>`
  - `rand()`
    - Produces a number between `0` and `RAND_MAX`
  - `srand( int )`
    - Seeds (initializes) the random number generator
- `<time.h>`
  - `time( NULL );`
    - Returns the current number of seconds since Jan 1, 1970

```
#include <stdlib.h>
#include <time.h>
…
    srand( time( NULL ) );
…
    int r = ( rand() % 100 ) + 1;   /* 1…100 */
```


The smallest C program to print the biggest prime number

Here it is (479 bytes):

```
int m=754974721,N,t[1<<24],a,*p,i,e=30295789,j,s,b,c,U;f(d){for(s=1<<23;s;s/=2,d=
d*1LL*d%m)if(s<N)for(p=t;p<t+N;p+=s)for(i=s,c=1;i;i--)b=*p+p[s],p[s]=(m+*p-p[s])
*1LL*c%m,*p++=b%m,c=c*1LL*d%m;for(j=0;j<N-1;){for(s=N/2;!((j^=s)&s);s/=2);if(++i
<j)a=t[i],t[i]=t[j],t[j]=a;}main(){*t=2;U=N=1;while(e/=2){N*=2;U=U*1LL*(m+1)/2%
m;f(362);for(p=t;p<t+N;)*p++=*p*1LL**p%m*U%m;f(41502754O);for(a=0,p=t;p<t+N;)a+=
*p<<(e&1),*p++=a%10,a/=10;}while(!*--p);t[0]--;while(p>=t)printf("%d",*p--);}
```

This program computes $2^{24036583}-1$, which is the biggest known prime number (more than 7 million digits!). For more information about how it was found and who found it, look at the GIMPS Project .

I compiled it successfully with gcc with i86 Linux. It takes about 2 minutes on a 2.4 GHz Pentium 4. In order to compile it, your C compiler must support the 64 bit long long type.

This program basically converts from base 2 to base 10. It is a non trivial task because it deals with numbers of millions of digits. The usual method (with repeated divisions by 10^N) would be far too slow. So I decided to use an Integer Fast Fourier Transform. I believe it is one of the smallest implementation of such an algorithm.

A previous version of this program to compute $2^{6972593}-1$ won the International Obfuscated C Code Contest of Year 2000.

This program is Freeware.
Fabrice Bellard - http://bellard.org/
last update: Jun 15, 2004