Computer Programming

Library

Hung-Yun Hsieh
November 1, 2022

Computer Programming

Standard Library

Standard Library Functions

- The *C++ Standard Library* is divided into many portions, each with its own header file
  - The header files contain the function prototypes (declarations) for related functions that form portions of the library (may also contain related class types and constants)
  - A header file "instructs" the compiler on how to interface with library and user-written components
  - ☞ The actual "content" (binary code) of the function is stored in the library file to be combined with the user-written codes during linking
  - ☞ Online resource: *http://www.cplusplus.com/reference/*

Working with Random Numbers

- `rand()` function
  - Generates unsigned integer between 0 and `RAND_MAX` (a pre-defined constant that is at least 32767)
  - Scaling and shifting to adjust the range
    - number = shiftingValue + `rand()` % scalingFactor
  - Fractional random number?
- Pseudorandom numbers
  - Long runs of numbers with good "random" properties
  - The run (sequence) may eventually repeat itself
  - Same sequence is generated under the same condition
  - ☞ To get different sequences, supply seed values to the random number generator

Seeds

The function `srand()` needs one integer argument and it does not return any value to the caller

- Linear congruential generator (LCG)
  - Random sequence X

$$X_{n+1} = (aX_n + b) \mod (m)$$

    - X_0: the seed or start value
    - Typical values used (*glibc*)

Provided that the offset b is nonzero, the LCG will have a full period (m) for all seed values if and only if
1. b and m and are relatively prime
2. a-1 is divisible by all prime factors of m
3. a-1 is a multiple of 4 if m is a multiple of 4

m (modulus)	a (multiplier)	b (increment)	output of rand()
2^{31}	1103515245	12345	Bits 30…0

- Seed value
  - Initialize the random sequence
  - Same seed will give the same sequence
  - Use function `srand(seed)` to set seed
    - Called before `rand()` is used to set the seed

Random Number Seeds

```
#include <iostream>
#include <cstdlib>
#include <iomanip>
using namespace std;

int main()
{
    int seed;
    cout<<"Preset random sequence:\t";
    for (int i=1;i<=10;i++) cout<< setw(3) <<rand()%10+1;
    cout<<endl;
    while (1) {
        cout<<"\nEnter a new seed: ";
        cin>>seed;
        srand(seed);
        cout<<"\nNew random sequence:\t";
        for (int i=1;i<=10;i++) cout<< setw(3) <<rand()%10+1;
    }
}
```

Header file for `rand()` and `srand()`

The `time()` Function

C++ allows the definition of new data types based on existing data types using `typedef`

- Calendar time
  - `time(0)`: returns the current time (measured in seconds) since 00:00:00 GMT, January 1, 1970
    - The argument to function `time()` is a pointer to a memory location for storing the time value retrieved (an integer type redefined as `time_t`)

```
typedef unsigned long time_t;
```

```
time_t time(time_t* ptr);
```

    - Different functions can be used to further process the returned value (`time_t`) for showing the more readable calendar time
  - Use the current time to set the seed

```
srand(time(0));
```

    - Different seeds/sequences if program runs at different times

More on `time()`

- Time structure `struct tm`
  - A structure (defined through keyword `struct`) is a *user-defined data type* created to group (like or unlike) data types as its member
  - The new structure `tm` contains members to hold the calendar date and time

```
struct tm {
   int tm_sec;         /* seconds,  range 0 to 59          */
   int tm_min;         /* minutes, range 0 to 59           */
   int tm_hour;        /* hours, range 0 to 23             */
   int tm_mday;        /* day of the month, range 1 to 31  */
   int tm_mon;         /* month, range 0 to 11             */
   int tm_year;        /* The number of years since 1900   */
   int tm_wday;        /* day of the week, range 0 to 6    */
   int tm_yday;        /* day in the year, range 0 to 365  */
   int tm_isdst;       /* daylight saving time             */
};
```

  - Function `localtime()` transforms from `time_t` to structure `tm`

```
tm* localtime(const time_t* ptr);
```

  - Function `strftime()` can further transform from `tm` to a character string depending on the specified flags

An Example

Note that **tm_sec** may go as high as 61 to allow for up to two leap seconds

```
#include <ctime>
#include <cstdio>

int main ()
{
    time_t now;
    tm * timeinfo;
    const char *wday[]={"Sun", "Mon", "Tue", "Wed", "Thu", "Fri", "Sat"};

    time (&now);
    timeinfo = localtime(&now);

    printf("Time: %u:%u:%u\n", timeinfo->tm_hour,
                               timeinfo->tm_min,
                               timeinfo->tm_sec);
    printf("Date: %u/%u/%u\n", timeinfo->tm_year+1900,
                               timeinfo->tm_mon+1,
                               timeinfo->tm_mday);
    printf("Day of Week: %s\n",wday[timeinfo->tm_wday]);
}
```

```
struct tm {
    int tm_sec;    /* seconds after the minute (0 to 61) */
    int tm_min;    /* minutes after the hour (0 to 59) */
    int tm_hour;   /* hours since midnight (0 to 23) */
    int tm_mday;   /* day of the month (1 to 31) */
    int tm_mon;    /* months since January (0 to 11) */
    int tm_year;   /* years since 1900 */
    int tm_wday;   /* days since Sunday (0 to 6) */
    int tm_yday;   /* days since January 1 (0 to 365) */
    int tm_isdst;  /* Daylight Savings Time (1: DST, 0: no) */
};
```

More on `printf()`

- The `printf` function with output formatting
  - ☞ Variable-length argument
  - Use `%` to start a *format sub-sequence*

http://www.cplusplus.com

Specifier	Corresponding argument
d	Signed decimal integer
u	Unsigned decimal integer
x	Unsigned hexadecimal integer
f	Decimal floating point
e	Floating point in scientific notation (mantissa/exponent)
g	Floating point using the shortest representation: %e or %f
c	Character
s	String of characters

```
%+02d
%8.3f
%-25s
%%
```

Support for user-defined (new) data type?
-- `cout`

- Additional sub-specifiers can be given

`%[flags][total_width][.precision][length]specifier`

`+-#0`

`l, ll, L, h, …`

Calendar Time

You can use either `struct tm` or `tm` to refer to the new data type

```
#include <iostream>
#include <ctime>

int main ()
{
    time_t now;
    tm * timeinfo;
    char buffer [80];

    time (&now);
    timeinfo = localtime(&now);

    strftime(
      buffer,
      sizeof(buffer),
      "Now it's %I:%M:%S%p.",
      timeinfo);

    std::cout << buffer << '\n';
}
```

Specifier	Replaced By	Example
%a	Abbreviated weekday name	Sun
%A	Full weekday name	Sunday
%b	Abbreviated month name	Mar
%B	Full month name	March
%c	Date and time representation	Sun Aug 19 02:56:02 2012
%d	Day of the month (01-31)	19
%H	Hour in 24h format (00-23)	14
%I	Hour in 12h format (01-12)	05
%j	Day of the year (001-366)	231
%m	Month as a decimal number (01-12)	08
%M	Minute (00-59)	55
%p	AM or PM designation	PM
%S	Second (00-61)	02
%U	Week number with the first Sunday as the first day of week one (00-53)	33
%w	Weekday as a decimal number with Sunday as 0 (0-6)	4
%W	Week number with the first Monday as the first day of week one (00-53)	34
%x	Date representation	08/19/12
%X	Time representation	02:50:06
%y	Year, last two digits (00-99)	01
%Y	Year	2012
%Z	Timezone name or abbreviation	CDT
%%	A % sign	%

The `clock()` Function

- Clock ticks
  - `clock()` returns the processor time consumed by the program
  - The value returned is measured in clock ticks (an integer type redefined as `clock_t`)

```
typedef unsigned long clock_t;
```

  - `clock()` is typically used *in pairs* (before & after some statements) to count the time elapsed between two calls
  - Divide the returned value by `CLOCKS_PER_SEC` to convert it to seconds
    - `CLOCKS_PER_SEC` is a constant equal to the number of ticks contained in one second

Execution Time

```
#include <iostream>
#include <cmath>
#include <ctime>
using namespace std;

int num_of_primes (int n) {
    int i,j,freq=n-1;
    for (i=2; i<=n; ++i) for (j=sqrt(i);j>1;--j)
    if  (i%j==0) {--freq; break;}
    return freq;
}

int main ()
{
    clock_t t = clock();
    cout << "Number of primes <=100,000 is " << num_of_primes(99999);
    t = clock() - t;
    cout << "\nCalculation time: " << t << " ticks ("
         << ((float)t)/CLOCKS_PER_SEC << " seconds)" << endl;
}
```

Divide `i` by each integer `j` that is greater than 1 and less than or equal to the square root of `i`

If the result of any of these divisions is an integer, then `i` is not a prime; otherwise it is a prime

String Functions

All functions are declared in <cstring>

```
typedef unsigned int size_t;
```

Function prototype	Function description
`size_t strlen( const char *s );`	Determines the length of string s. The number of characters preceding the terminating null character is returned.
`char *strcpy( char *s1, const char *s2 );`	Copies the string s2 into the character array s1. The value of s1 is returned.
`char *strcat( char *s1, const char *s2 );`	Appends the string s2 to s1. The first character of s2 overwrites the terminating null character of s1. The value of s1 is returned.
`int strcmp( const char *s1, const char *s2 );`	Compares the string s1 with the string s2. The function returns a value of zero, less than zero (usually −1) or greater than zero (usually 1) if s1 is equal to, less than or greater than s2, respectively.
`char *strtok( char *s1, const char *s2 );`	A sequence of calls to strtok breaks string s1 into "tokens"—logical pieces such as words in a line of text. The string is broken up based on the characters contained in string s2. For instance, if we were to break the string "this:is:a:string" into tokens based on the character ':', the resulting tokens would be "this", "is", "a" and "string". Function strtok returns only one token at a time, however. The first call contains s1 as the first argument, and subsequent calls to continue tokenizing the same string contain NULL as the first argument. A pointer to the current token is returned by each call. If there are no more tokens when the function is called, NULL is returned.

```
strncpy(s1, s2, n)
strncat(s1, s2, n)
strncmp(s1, s2, n)
```

Manipulating String

```
#include <iostream>
#include <cstring>
using namespace std;

int main( )
{
    char text[70];
    int occupied, reserved;

    strcpy(text, "This is an incomplete sentence, ");
    strcat(text, "but this is.");

    occupied = strlen(text);
    reserved = sizeof(text)/sizeof(char);
    cout << "string=" << text <<endl
         << "occupied=" << occupied << " reserved=" << reserved << endl;

    if (strcmp(text, "Another sentence.")==0)
    cout << text << " is equal to \"Another sentence.\"";
}
```

Be careful that enough memory is allocated at the destination pointer `text` to hold the source string

Null character is not included

Using `strtok()`

Note that the second argument to `strtok()` is a separator of **one character,** which can be changed in different calls

```
#include <iostream>
#include <cstring>
using namespace std;

int main( )
{
    char sentence[] = "This is a sentence with 7 tokens";
    char *tokenPtr;

    cout << "The string to be tokenized is:\n" << sentence
         << "\n\nThe tokens are:\n\n";

    tokenPtr = strtok(sentence, " ");
    while (tokenPtr != NULL)
    {
        cout << tokenPtr << '\n';
        tokenPtr = strtok(NULL, " ");
    }
    cout << "\nAfter strtok(), sentence=" << sentence << endl;
}
```

T	h	i	s		i	s		a		s	e	n	t	e	n	c	e		w	i	t	h		7		t	o	k	e	n	s	\0

Multiple calls to `strtok()` are necessary to break a string into multiple tokens

Some Handy Functions

- Conversion from string to numbers *<cstdlib>*
  - `atof()`, `atoi()`, `atol()`, `atoll()`: conversion from string to double, integer, long, and long long respectively

```
double atof(const char*);
int atoi(const char*);
```

  - More sophisticated version: `strof()`, `strol()`, …
- Program termination *<cstdlib>*
  - `exit()` terminates the program anywhere it is called

```
void exit(int status);
```

EXIT_SUCCESS or EXIT_FAILURE

  - `atexit()` sets function to be executed on exit

```
void atexit(void (*func)(void));
```

Function pointer

Using `atexit()`

```
#include <iostream>
#include <cstdlib>
using namespace std;

void fnExit1() { cout << "Exit function 1" << endl; }
void fnExit2() { cout << "Exit function 2" << endl; }

int main( )
{
    atexit(fnExit1);
    atexit(fnExit2);

    cout << "Main program" << endl;
}
```

If more than one `atexit` function is specified to register multiple functions, they are executed in reverse order

```
Main program
Exit function 2
Exit function 1
```

More on Function Pointers

```
#include <iostream>
using namespace std;

int myadd(int a, int b)             { return a+b; }
int myminus(int a, int b)           { return a-b; }
int mymultiply(int a, int b)        { return a*b; }
int mydivide(int a, int b)          { return a/b; }

int calc_do(int x, int y, int (*opt)(int, int)) { return opt(x, y); }

int main( )
{
    int a = 3, b = 2;
    int (*p)(int, int) = myadd;
    cout << a << "+" << b << "=" << calc_do(a, b, p) << endl;
    p = myminus;
    cout << a << "-" << b << "=" << calc_do(a, b, p) << endl;
    cout << a << "*" << b << "=" << calc_do(a, b, mymultiply) << endl;
    cout << a << "/" << b << "=" << calc_do(a, b, mydivide) << endl;
}
```

Recall Pointers to Static Arrays

- Pointer to static array

```
int a[2][3] = {{1, 2, 3}, {4, 5, 6}};
int (*p)[3];
p = a;
```

`p` is of type `int (*)[3]`

`int *q[3]` is an array of 3 elements that have type `int *`

```
void printArray(int (*a)[3], int size);
printArray(p, 2);
```

- Pointer to function

```
int myadd(int, int);
int (*p)(int, int);
p = myadd;
```

`p` is of type `int (*)(int, int)`

`int *q(int, int)` is a function that returns type `int *`

```
int calc_do(int x, int y, int (*opt)(int, int));
calc_do(3, 2, p);
```

More on Function Pointers (Take 2)

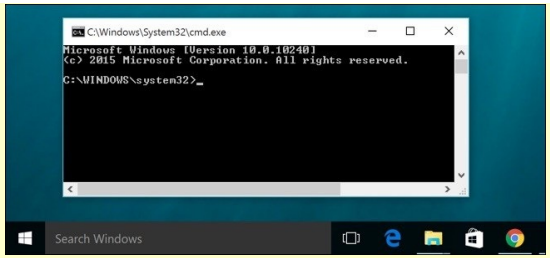
```
#include <iostream>
using namespace std;

int myadd(int a, int b)             { return a+b; }
int myminus(int a, int b)           { return a-b; }
int mymultiply(int a, int b)        { return a*b; }
int mydivide(int a, int b)          { return a/b; }

int calc_do(int x, int y, int (*opt)(int, int)) { return opt(x, y); }
int (*calc_arr[])(int, int) = { myadd, myminus, mymultiply, mydivide };
const char* calc_op[] = { "+", "-", "*", "/" };

int main( )
{
    int a = 3, b = 2;

    for (int i=0;i<=3;i++)
        cout << a << calc_op[i] << b << "=" <<
                    calc_do(a, b, calc_arr[i]) << endl;
}
```

More on Function Pointers (Take 3)

```
#include <iostream>
using namespace std;

typedef int (*calc_func)(int, int);
int myadd(int a, int b)             { return a+b; }
int myminus(int a, int b)           { return a-b; }
int mymultiply(int a, int b)        { return a*b; }
int mydivide(int a, int b)          { return a/b; }
int calc_do(int x, int y, calc_func opt) { return opt(x, y); }
calc_func calc_arr[]  = { myadd, myminus, mymultiply, mydivide };
const char* calc_op[] = { "+", "-", "*", "/" };

int main()
{
    int a = 3, b = 2;

    for (int i=0;i<=3;i++)
        cout << a << calc_op[i] << b << "=" <<
                    calc_do(a, b, calc_arr[i]) << endl;
}
```

More on Function Pointers (Take 4)

```
#include <iostream>
using namespace std;

typedef int (*calc_func)(int, int);
int myadd(int a, int b)            { return a+b; }
int myminus(int a, int b)          { return a-b; }
int mymultiply(int a, int b)       { return a*b; }
int mydivide(int a, int b)         { return a/b; }
int calc_do(int x, int y, calc_func opt) { return opt(x, y); }
calc_func calc_arr[]  = { myadd, myminus, mymultiply, mydivide };
const char* calc_op[] = { "+", "-", "*", "/" };

int main(int argc, char* argv[])
{
    int a = 3, b = 2;
    if (argc==2) for (int i=0;i<=3;i++)
    if (strcmp(argv[1], calc_op[i])==0)
        cout << a << calc_op[i] << b << "=" <<
                    calc_do(a, b, calc_arr[i]) << endl;
}
```



command prompt

Command line options:
`argv[0]` is the program itself,
`argv[1]` is the first argument, …
`argc` indicates the array length