Programming in C/C++
Lecture 2

Coordinator: Dr. Natalia Silvis-Cividjian (nsilvis@few.vu.nl)
Lecturers: Dr. Jeffrey Napper

vrije Universiteit amsterdam

Check announcements on the C/C++ website:
- /usr/bin/g++ on kits.few.vu.nl

Questions?
- Previous lecture?
- Homework?
  - pointers are usually unsigned long

Outline
- Input, output, and files
- Linked lists

Input, output, and files
- Standard input and output (I/O)
- File I/O
- Read more in input, output, and files in Ch. 17 in Prata C++ primer

Input and output (I/O)
- C/C++ do not have built-in I/O functions
  - Like Java, I/O is included in standard library
- I/O is left to compiler implementers
- C uses a standard library of functions: standard I/O package, stdio.h
- C++ uses standard libraries of classes: iostream.h and fstream.h

<stdio.h>
Standard I/O library inherited from original C. Requires #include <stdio.h>
- How to read/write a character? char c = getchar(); putchar('x');
- How to read/write a C-string? gets(name); puts("Hello");
- How to format input/output? scanf and printf using conversion characters:
  - Read manual on functions for the many details (and fscanf, sscanf, fprintf, sprintf...)

scanf("%d", &a); // reads an integer from keyboard
int i = 5; int j = 7;
double d = i / (double) j;
printf("the value of \%d over \%d is \%lf", i, j, d); // writes on the screen
C++ streams

A stream is a flow of bytes (~river).

Each program automatically has an input stream and an output stream.

A C++ stream is an object with member functions:
- open(), close(), fail(), getline(), get(), put(), setf(), seek()

Streams can be arguments for functions, but must be called by reference.

Streams and buffers

- I/O is handled efficiently using a buffer:
  - A block of memory used as temporary storage for the transfer of information between 2 devices
  - C++ flushes the input buffer when ENTER is pressed
  - Why your program won't read a single char!
  - C++ flushes the output buffer when NL is reached
  - Why your program won't write a single char!

Buffers

- Store output data for next process
  - Ex: people on train platform
- Collect data from a slower process
  - Ex: internet video streaming
- Reduce the need for synchronization
  - Ex: fewer calls to rewrite screen

Console I/O <iostream>

- `cin` is a standard input stream connected to the keyboard.
  - No need to declare.
  - Example: `cin >> number;
- `cout` is a standard output stream connected to the screen.
  - No need to declare.
  - Example: `cout << "the number is" << number;`

I/O class inheritance

- `cin` is an object of istream class.
- `cout` is an object of ostream class.

```cpp
#include<iostream> // creates 4 stream objects: cin, cout, cerr, clog
```
cin translates text from keyboard to other type
- cin ignores (skips) white spaces (space, tab, nl) This can be confusing!

Suppose the following input: 38.5 19.2

char ch; cin >> ch; // will read ch = 3
int n; cin >> n; // will read n=38
double x; cin >> x; // will read x = 38.5

What is wrong with this code?

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

int main()
{
    int ct = 0;
    char ch;
    cin >> ch;
    while (ch != '
') // Can this be false?
    {
        cout << ch;
        ct++;
        cin >> ch;
    }
    cout << ct << endl;
    return 0;
}
```

Use get() to check for NL

- Note that newline (NL) can be different on different platforms: ‘\n’ (Unix) or ‘\n\r’ (Windows)

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

int main()
{
    char ch;
    cin.get(ch);
    while (ch != '\n')
    {
        cout << ch;
        cin.get(ch);
    }
    return 0;
}
```

C-strings input

- char name[20];
- char dessert[30];
- cin >> name;
- cin >> dessert;

Problem: cin is a word-oriented method. cin reads until the first whitespace. You cannot input “Alan Pearson” for name.

Solution: use getline – line oriented method
C-strings input

- Line oriented methods read line until NL
- Here getline() function is a class method for istream class. cin is a istream object.
- getline() discards the NL!
- get() leaves NL in the input queue

```cpp
#include <iostream>
char name[20];
char dessert[30];
cin.getline(name); // beware overflow!
cin.getline(dessert);
```

C++ string input

- Using cin:
  ```cpp
  string line ;
cin >> line ; // reads in a string until a whitespace, ignores leading spaces
  ```

C++ strings input

- Use getline. Reads a line until a NL into a C++ string:
  ```cpp
  string str;
generate(cin,str); // notice different syntax
  ```

- Here getline is not a class method because istream class cannot process string. It was designed long before string existed.
- This method has a bug in Visual C++. See website.

C++ string console input

- ```cpp
  getline (cin, line, ch) // reads a string line until ch. Also buggy in Visual C++
  ```

Numbers console input

- How to read more integers from the keyboard?
  Use a sentinel-controlled while loop. A sentinel (or trailer) is a special predetermined value to mark the end of input.

```cpp
const SENTINEL = -999 ;
int data ;
cout << "Enter an integer (" << SENTINEL
 << " to end input): " ;
cin >> data ;
while (data != SENTINEL)
{ 
do_stuff (data) ;
cin >> data ;
}
```

Integers input

- How to check for valid input? cin stream returns FALSE if the input operation fails (e.g., a char is fed instead of integer)

```cpp
int data ;
if (!(cin >> data))
{ 
cout << "please enter only numeric characters" ;
cin.clear() ;
cin.ignore (1000) ;
}```
Mixed string and numeric input

#include <iostream>
using namespace std;
char address[80];
int years;
int main()
{
    cout << "How old are you?";
    cin >> years;
    cout << "What is your address?";
    cin.getline(address, 80);
    return 0;
}

PROBLEM!

#include <iostream>
using namespace std;
char address[80];
int years;
int main()
{
    cout << "How old are you?";
    cin >> years;
    cin.get() ; // reads the NL after years
    cout << "What is your address?";
    cin.getline(address, 80);
    return 0;
}

Console output

• How to write one character to the screen?
cout << ch;
or
cout.put (ch) // sends one character to the screen.
Cout(’W’) writes 87 in older versions of C++.
Cout.put(’W’) works fine.

Formatting with cout

• Include this sequence
  cout.setf (ios::fixed);
  cout.setf (ios::showpoint);
  cout.precision (2); // floating point display precision

Or use a header additional manipulators

#include <iomanip>.

cout << setw (7) << number; // with iomanip

cout.fill('*'); // replaces filling spaces with *

File I/O
#include <fstream>

Files are also streams like cin and cout. <fstream> library contains a few classes:

- `ifstream` for input file stream;
  
  Example: `ifstream in_stream ;`

- `ofstream` for output file stream;
  
  Example: `ofstream out_stream ;`

File names

An external name (SYSTEM name) = "infile.dat"

The stream name (PROGRAM VARIABLE) = `in_stream`.

File I/O

Steps to read from a file `infile.dat` and write to a file `outfile.dat`:

1. Place the `#include <fstream>` directive
2. Declare input and output streams
3. Connect each stream to a file = open files
4. Get input from the file
5. Send output to the output file
6. Disconnect the streams = close files

```cpp
#include <fstream>
using namespace std;

ifstream in_stream ;
ofstream out_stream ;

int var;
in_stream.open("infile.dat");
out_stream.open("outfile.dat");
in_stream >> var ;
out_stream << var ;
in_stream.close();
out_stream.close();
```

File I/O: useful functions

- How to read the file name from keyboard?

  ```cpp
  string fileName ;
  ifstream inStream ;
  cin >> "Enter file name: \n" ;
inStream.open (fileName.c_str()) ; // converts C++ string to C-string
  ```

- How to read integers from a file?

  ```cpp
  in_stream >> number ; // jumps over blank characters in the file.
  ```

Fail()

- How to check for file opening success? Use `fail()` member function and `exit()` from <cstdlib> library.

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

int main() {
    ifstream ins ; // file to be opened for input
    ins.open ("data.dat");
    if (ins.fail()) // if opening fails, end program
        cout << "input file opening failed. \n" ;
    exit (1) ; // exit function from cstdlib library
    }
```
How to test the end of the file?
Each stream has a member function `eof()` which becomes true when the program tries to read a character beyond the end of the file.

```cpp
in_stream.get( next )
while (! in_stream.eof()) {
    do_stuff( next );
    in_stream.get( next );
}
```

Example:
```
abc
```
- program reads:
  - `eof()`: false
  - `a`: false
  - `b`: false
  - `c`: false
End of file marker: true => stop

Other solutions
- Use `peek()`
- Use `cin.get()` (returns zero if reaches EOF).

File I/O: `eof()` and `fail()` example
```cpp
#include <iostream>
#include <fstream>
#include <cstdlib>
using namespace std;

int main() {
    float x, sum;
    ifstream ins;  //input stream
    ofstream outs;  //output stream
    ins.open( "indata.dat" );  //open files, exit if fail
    if (ins.fail()) {
        cout << "Can't open indata.dat" << endl;
        exit(1);
    }
    outs.open( "results.dat" );
    if (outs.fail()) {
        cout << "Can't open results.dat" << endl;
        exit(1);
    }
    sum = 0.0;
    ins >> x;
    while (!ins.eof()) {
        sum += x;
        ins >> x;
    }
    outs << "The sum is " << sum << endl;
    cout << "The sum is " << sum << endl;
    ins.close();
    outs.close();
    return 0;
}
```

Outline
- Input, output and files
- Linked lists

Reading
Read from chapter 12, Prata about linked lists
- W. Savitch, Problem solving with C++, Addison Wesley, 2005
- Liberty Jones, Teach yourself C++ in 21 days, SAMS, 2005
A (singly) linked list is a dynamic data structure built from nodes which are connected by pointers.

Each node has always 2 fields:
- data (struct or class)
- a pointer to the next node

A list should always have:
- a head = a pointer to the first node
- an end-of-list marker = NULL

**About NULL**
- NULL is an invalid pointer! (actually zero)
- NULL is used to detect the end of a list
- included in different library headers: `<iostream>`, `<cstddef>`
- use:
  - `p = NULL;`
  - pointer `p` points to nothing, where `p` can be any type of pointer

**Struct Node**
```
struct Node {
    string article_name;
    int quantity;
    string unit;
    Node *next;
};
```

```
Node *head; // head is a pointer to a Node

(*head).article_name = "Potato";
(*head).quantity = 50;
head->article_name = "Potato";
head->quantity = 50;
```

**Accessing Node Data**

**Linked lists manipulation**

How to:
- start a list
- insert a node at the head of the list
- search a linked list for a target
- insert a node at location inside a list
- remove a node from a list
### How to start?

```c
struct Node {
    int data;
    Node *next;
};

Node *head;
head = new Node;
head->data = 20;
head->next = NULL;
```

**Action:** Start a list of integer numbers

### Tip: use typedef

```c
define a type named NodePtr as a pointer to a Node
```

```c
NodePtr head;
```

#### Definition

```
typedef <old type name> <new type name>
```

**Purpose:** Defines a new type by renaming an existing type.

**Tip:** Use typedef.

### Insert a node at the head of a list

**Action:** Insert number 12 at the head of the list

```c
void head_insert (NodePtr& head, int the_number) {
    NodePtr temp_ptr;
    temp_ptr = new Node;
    temp_ptr->data = the_number;
    temp_ptr->next = head;
    head = temp_ptr;
}
```

### Insert at the head of a list: the code

**Action:** Insert number 12 at the head of the list

```c
head = head_insert (head, 12);
```

### Searching a linked list

**Problem:** Design a function to search a target in a linked list.

**The function returns a pointer to the first node that contains the target.** If the target is not found, the function will return NULL. (Example: look for a book in a library on a messy shelf)

**Solution:**

1. The only way to move through a list is to follow the pointers.
2. We need a pointer to move through the list = iterator. Let’s name it here.
3. Here starts from the head node.
4. While here is not pointing to a node containing the target and here is not the last node, here jumps to the next node.
5. If the node pointed by here has the target, return here.
6. Else, return NULL.

```c
NodePtr here = head;
while (here != NULL) {
    if (here->data == target) {
        return here;
    }
    here = here->next;
}
return NULL;
```
Search a list

Action: Search for number 15

Search a list : code (while)

```c
NodePtr search (NodePtr head, int target)
{
    NodePtr here = head ;
    if (NULL == here) { return NULL ; }
    while (here->data != target && here->next != NULL)
        here = here->next ; // iterate
    if (here->data == target)
        return here ;
    else
        return NULL ;
}
```

Search a list : code (for)

```c
NodePtr search (NodePtr head, int target)
{
    if (NULL == head) { return NULL ; }
    for( NodePtr here = head ;
        NULL != here ;
        here = here->next ) {
        if (here->data == target)
            return here ;
    }
    return NULL ;
}
```

Insert at location

Action: insert number 40 after the node pointed by after_me

Insert at location

```c
void insert (NodePtr after_me, int the_number)
{
    NodePtr temp_ptr = new Node ;
    temp_ptr->data = the_number ;
    temp_ptr->next = after_me->next ; // #1
    after_me->next = temp_ptr ; // #2
}
```

Remove a node from a list

```
use two help pointers:
before points to the node before the one to be deleted
discard points to the node to be removed
before->next = discard->next ; // #1
delete discard ; // #2
```

```c
NodePtr search (NodePtr head, int target)
{
    NodePtr here = head ;
    if (NULL == here) { return NULL ; }
    while (here->data != target && here->next != NULL)
        here = here->next ; // iterate
    if (here->data == target)
        return here ;
    else
        return NULL ;
}
```
Applications of linked lists

Push(A); Pop();

Enqueue(A); Dequeue();

Stack operations

void Push(NodePtr head, int elem) {
    NodePtr node = new Node;
    node->data = elem;
    node->next = head;
    head = node;
}

NodePtr Pop(NodePtr head) {
    NodePtr node = head;
    if (NULL != head) {
        head = node->next;
        return node;
    }
}

Queue operations

void enqueue(NodePtr& head, int elem) {
    NodePtr node = new Node;
    node->data = elem;
    node->next = head;
    head = node;
}

NodePtr dequeue(NodePtr& head) {
    if (NULL == head) return NULL;
    NodePtr prev = NULL;
    NodePtr last = head;
    while(NULL != last->next) {
        prev = last;
        last = last->next;
    }
    if (NULL != prev) {
        prev->next = NULL;
    } else { // List has 1 node!
        head = NULL;
    }
    return last;
}

Questions?