CS2311 Computer Programming

MS Y. MONG

Review

About the Course

- Lecturer
  - Ms. Y MONG,
    - YEUNG(AC1)Y6415, 3442 8503, csymong@cityu.edu.hk
- TAs
  - Offer general help on exercises and software setup during labs
- Labs
  - 2-hour "hands-on" practice in CSC labs
  - Analyzing simple problems and implementing computer programs

Assessment

- Coursework (40%)
  - One Quiz (15%), week 7
    - One question is going to be very similar to an exercise from the labs.
  - Assignments: (15%)
    - analyze more challenging problems
    - implement and present solutions
  - Lab Exercises (7%)
    - we'll randomly take 1 exercise from 7/12 labs to mark
    - Deadline is 2 hours after your lab session
  - Lecture Attendance (3%)
    - attend more than 9 lectures
    - to encourage continuous learning

About the Course – Course Outcomes

1. Explain the structure of an object-oriented computer program;
2. Analyze, test and debug computer programs;
3. Solve a task by applying effective programming techniques, which involve advanced skills like using dynamic data structures;
4. Design and construct well-structured programs with good programming practices.

About the Course – Resources

- Course website on Canvas
  - Lecture slides
  - Lab notes
  - Assignments
  - Announcements, etc.
- Microsoft Visual Studio 2015 (Windows)
  - For compiling & debugging programs
  - Can be installed on your Windows machines
  - *Your best friend for this course*
  - More info in Lab 1, including Mac/Linux alternatives
- PASS (Program Assignment aSsessment System)
  - Program testing and submission
  - For labs & assignments

About the Course – Course Schedule

Wk	Lecture Topic	Tutorial Topic	Assessment
1	Introduction, simple programs	Intro to VS2013	
2	The C++ programming language, operators, data Types	Simple programs & PASS	
3		Simple programs & operators	
4	– N/A –		
5	Flow control (if, switch)	Flow control (if, switch)	
6	Flow control (for, while)	Flow control (for, while) Intro to VS Debugger	Assign. 1 Due
7	Arrays (1D and 2D)	Arrays	Mid-Term Test
8	Functions		
9	Class and Object	Class and object	Assign. 2 Due
10	Pointers (pass by ref)	Pointers (pass by ref)	
11	Pointers (arrays)	Pointers (arrays)	
12	Strings	Strings	
13	File I/O, Other topics (if time) Revision	File I/O, Other topics (if time), revision	Assign. 3 Due

Basic Program

- A simple C++ program will have

```
#include <iostream>        //A preprocessor directive
using namespace std;       //namespace declaration
int main() {
    /* the starting point of program execution  */

    return 0;
}
```


Mixed Use of Local and Global Variables

```
#include <iostream>
using namespace std;
int number1=12;
int number2=3;
/* Any code after this line can access these global variables */
int main() {
  int result;
  result = number1+number2;
  return;
} /* You can't access "result" beyond this line */
```

global

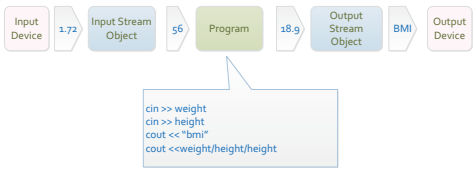
local

Keywords (reserved words) – covered in this course

Data type	char	double	float	int	bool
	long	short	signed	unsigned	void
Flow control	if	else	switch	case	
	break	default	for	do	
	while	continue			
Others	using	namespace	true	false	sizeof
	return	const	class	new	delete
	operator	public	protected	private	friend
	this				

Basic I/O – cin and cout

- C++ comes with an **iostream** package (library) for basic I/O.
- **cin** and **cout** are objects defined in **iostream** for keyboard input and screen display respectively
- To read data from **cin** and write data to **cout**, we need to use extraction/input operator (**>>**) and insertion/output operator (**<<**)



Programming Styles

- Programmers should write code that is understandable to other people as well
- Meaningful variable names
- Which is more meaningful

  tax = temp1*temp2; // not meaningful

  tax = price*tax_rate; // good
- Meaningful Comments
  - Write comments as you write the program
- Indentation

C++ predefined data types

- Numerical
  - **int**: Integers (1, 1743, 0, -45)
  - **float**, **double**: real numbers (0.25, 6.45, 3.01e-5)

    ```
    float x;
    double z=1.0;
    ```
- Character
  - **char**: a single ASCII character (a, e, o, \n)

    ```
    char c;
    ```
- Logical
  - **bool**: Boolean (true, false)

    ```
    bool b;
    ```
- Other
  - **void** : empty values

int

- Typically, an **int** variable is stored in four bytes (1 byte = 8 bits).
- The length of an **int** variable restricts the range of values it can store, e.g., a 32-bit **int** can store any integer in the range of -2^{31} and $2^{31}-1$, i.e. -2147483648 to 2147483647
- When an **int** is assigned a value greater than its maximum value, overflow occurs and this gives illogical results; similarly underflow may occur when a value smaller than the minimum value is assigned. However, C++ does **NOT** inform you the errors.

float and double

- **float**, **double** and **long double** are used to represent real numbers using the floating point representation.

  ```
  double weight = 120.82;
  ```
- **float** uses less memory (4 bytes), but is less accurate (7 digits after decimal point); **double** uses more memory (8 bytes) but more accurate (15 digits after decimal point)
- We use **double** most of the time. It's also the default type for floating numbers in C++.
- Exponent representation is also acceptable,
  - e.g., 1.23e2 and 3.367e-4 (1.23×10^2, and 3.367×10^{-4})

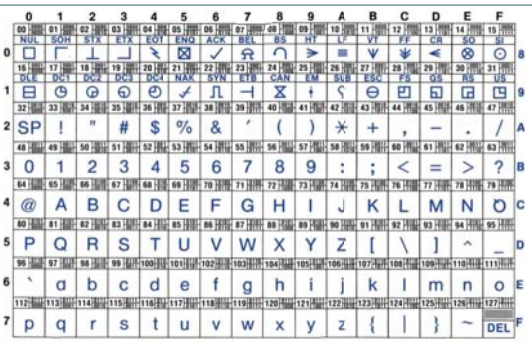
    ```
    double weight = 1.23e2;
    ```


Data Type char

- Every character is represented by a code
  - American Standard Code for Information Interchange (ASCII)
- Used to store a **single** ASCII character, enclosed by the **single** quotation mark

  ```
  char c = 'a';
  char c = '\n';
  ```
- Characters are (almost the same as) integers
- Characters are treated as small integers, and conversely, small integers can be treated as characters
- A character takes one byte
  - $2^8 = 256$, small integers
  - Internally, 'a' is stored as the following bit pattern 0 1 1 0 0 0 1
  - It is equivalent to an integer 97

ASCII Code



Strings (cstring)

- A string is a sequence of characters.
  - A **string** is treated as an array of characters. We call it **cstring**. (Another type of string is a **String** object)
- Strings are delimited by double quotation marks "", and the identifier must be followed with []

  char name[] = "John Doe";
- Remember escape sequences?

  char name[] = "\"hello\n\"";
- To extend a string beyond one line, use backslash \

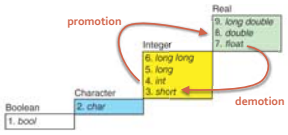
  char name[] = "Alex looooooong \
  John";

Type Conversion

- **Implicit** type conversion
  - binary expressions (e.g. x + y): lower-ranked operand is promoted to higher-ranked operand
  - assignment (e.g. x = y): right operand is promoted/demoted to match the variable type on the left
- **Explicit** type conversion (type-casting)

  example: **int** i = 10; **double** j = **(double)** i;

  Demoted values might **change** or become invalid



Constants

- Everything we covered before for variables apply for constants
  - type, name, scope
- Declaration format:

  const data_type variable/constant identifier = value;
- Examples:

  const float pi = 3.14159;
  const int maxValue = 500;
  const char initial = 'D';
  const char student_name[] = "John Chan";

Operators

Category	Examples
Arithmetic	+, -, /, *, %, =, ++, --
Comparison/relational	==, !=, >, <, >=, <=
Logical	!, &&, ||
Bitwise	~, &, |, ^, <<, >>
Compound assignment	+=, &=, <<=, etc.
Member and pointer	a[b], *, &, ->, etc.
Others	::, sizeof, etc.

Increment & Decrement Operators

- Increment and decrement operators: **++** and **--**
  - **k++** and **++k** are equivalent to **k=k+1**
  - **k--** and **--k** are equivalent to **k=k-1**
- Post-increment and post-decrement: k++ and k--
  - k's value is altered AFTER the expression is evaluated

  int k = 1, j;
  j = k++; /* result: j is 1, k is 2 */
- Pre-increment and pre-decrement: **++k** and **--k**
  - k's value is altered BEFORE evaluating the evaluation

  int k = 1, j;
  j = ++k; /* result: j is 2, k is 2 */

Examples of Assignment Statements

```
/* Invalid: left hand side must be a variable */
a + 10 = b;

/*assignment to constant is not allowed*/
2=c;

/* valid but not easy to understand */
int a, b, c;
a = (b = 2) + (c = 3);

/* avoid complex expressions*/
int a, b, c;
b = 2;
c = 3;
a = b + c;
```

Error!!

cout – Change the width of output

- Change the width of output
  - Calling member function width or using setw manipulator
  - Must **#include <iomanip>** for setw
  - Leading blanks are added to any value fewer than width
  - If formatted output exceeds the width, the entire value is printed
  - Effect last for one field only

Approach	Example	Output (◆ for space)
cout.width(*width*)	cout.width(10); cout << 5.6 << endl; cout.width(10); cout <<57.68 << endl;	◆◆◆◆◆◆◆5.6 ◆◆◆◆◆57.68

setw(*width*)	cout << setw(5) << 1.8;
	cout << setw(5) << 23 << endl;
	cout << setw(5) << 6.71;
	cout << setw(5) << 1 << endl;
		◆◆1.8◆◆◆23
		◆6.71◆◆◆◆1

Control Structures

- **if** statement
- Boolean logic
- **switch** statement

Examples – Single Statement

```
if (x > 5) {
    cout << "x is too large";
}
```

```
if (x > 5)
    cout << "x is too large";
else if (x < 3)
    cout << "x is too small";
```

```
if (x > 5)
    cout << "x is too large";
else if (x < 3)
    cout << "x is too small";
else
    cout << "x is a valid answer";
```


Short-circuit Evaluation

- Given integer variables i, j and k, what are the outputs when running the program fragment below?

```
k = (i=2) && (j=2);
cout << i << j << endl; /* 2 2 */
k = (i=0) && (j=3);
cout << i << j << endl; /* 0 2 */
k = i || (j=4);
cout << i << j << endl; /* 0 4 */
k = (i=2) || (j=5);
cout << i << j << endl; /* 2 4 */
```


Example

```
int x;
cin >> x;
switch (x) {
    case 0:
        cout << "Zero";
        break; /* no braces is needed */
    case 1:
        cout << "One";
        break;
    case 2:
        cout << "Two";
        break;
    default:
        cout << "Greater than two";
} //end switch
```


Loops

- **while** and **do while**
- **for**

while

```
#include <iostream>
using namespace std;
int main() {
    int    cnt = 0, n;
    float  max, x;

    cout << "The maximum value will be computed. \n";
    cout << "How many numbers do you wish to enter? ";
    cin >> n;
    while (n <= 0) { /* ensure a positive number is entered */
        cout << "\nERROR: Positive integer required. \n\n";
        cout << "How many numbers do you wish to enter? ";
        cin >> n;
    }
    cout << "\nEnter " << n << " real numbers: ";
    cin >> x; /* read 1st number */
    max = x;
    /* pick the largest number in while-loop */
    while (++cnt < n) {
        cin >> x; /* read another number */
        if (max < x)
            max = x;
    }
    cout << "\nMaximum value: " << max << endl;
    return 0;
}
```


Example of do .. while

```
bool error;
int n;
do {
    cout << "Input a positive integer: ";
    cin >> n;
    if (error = (n <= 0))
        cout << "\nError: negative input! \n";
} while (error);
...
```


Examples of for

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

int main() {
    int i;
    for (i=0; i<10; i++)
        cout << i << endl;
    return 0;
}
```

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

int main() {
    int i;
    for (i=0; i<10; i++) {
        if (i%2 == 0)
            cout << i << endl;
    }
    return 0;
}
```

Array Definition

Data type of the array elements

Size of the array

```
int mark[10];
```

Name of the array

There are ten elements in this array
mark[0], mark[1], ..., mark[9]

The ith array element is mark[i-1].

The range of the subscript i is from 0 to array_size-1

The location mark[10] is invalid. Array out of bound!

Counting digits

- Input a sequence of digits [0, 1, 2, ..., 9], which is terminated by -1
- Count the occurrence of each digit
- Use an integer array count of 10 elements
  - count[i] stores the number of occurrence of digit i


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

int main() {
    int count[10] = {0};      //number of occurrence of digits
    int digit;                //input digit
    int i;                    //loop index
    //read the digits
    do {
        cin >> digit;
        if (digit >= 0 && digit <= 9)
            count[digit]++;
    } while (digit != -1);    //stop if the input number is -1
    //print the occurrences
    for (i=0; i<10; i++)
        cout << "Frequency of " << i << " is " << count[i] << endl;
    return 0;
}
```

The Program

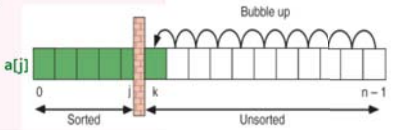
```
int main() {
    int array1[5] = {10, 5, 3, 5, 1};
    int array2[5];
    int i;
    bool arrayEqual=true;
    cout << "Input 5 numbers\n";
    for (i=0; i<5; i++)
        cin >> array2[i];
    for (i=0; i<5 && arrayEqual; i++)
        if (array1[i] != array2[i])
            arrayEqual = false;

    if (arrayEqual)
        cout << "The arrays are equal";
    else
        cout << "The arrays are not equal";
    return 0;
}
```

```
Input 5 numbers
10 5 3 5 1
The arrays are equal
```

```
Input 5 numbers
10 4 3 5 2
The arrays are not equal
```

Bubble Sort

```
int main() {
    const int n = 10;
    int a[n], tmp;
    cout <<"Input" << n <<" numbers: ";
    for (int j=0; j<n; j++)
        cin >> a[j];
    for(int j=0; j<n-1; j++)        // outer loop
        for(int k=n-1; k>j; k--)    // bubbling
            if (a[k]<a[k-1]) {
                tmp = a[k];         // swap neighbors
                a[k] = a[k-1];
                a[k-1] = tmp;
            }
    cout << "Sorted: ";
    for(int j=0; j<n; j++)
        cout << a[j];
    return 0;
}
```



Function Components

Function name

Input (Parameter/Argument)

Return type

```
void printHello (int n) {
    int i;
    for (i=0; i<n; i++)
        cout << "Hello\n";
}
```

Function body

n is defined as input,
therefore there is no need to declare n in the function body again

Parameters Passing: Call-by-Value

```
void f (int x) {
    x = 4;          // we modify the value x to 4
    // Do we modify y at the same time? NO
    y =4;           // syntax error: y is local to main
}
int main() {
    int y = 3;
    f(y);
    cout << y;      // print 3, y remains unchanged
    return 0;
}
```

The variables x and y are **local** variables.
y is local to **main** (), so we cannot use **y** in f().
x and y are two independent variables.
When x is modified, y will not be affected.

Parameters Passing: Arrays

The size of array is optional.
void f(int a[])

if the content of **a[i]** is modified in the function, the modification will persist even after the function returns (Call by reference)

```
void f(int a[]) {
    cout << a[0] << endl;     //1 is printed
    a[0] = 10;
}

int main () {
    int a[3] = {1, 2, 5};     //an array with 3 elements
    f(a);                     //calling f() with array a
    cout << a[0] << endl;     //10 is printed
    return 0;
}
```

Only need to input the array name!

Call-by-Reference

- Parameters pass to a function can be updated inside the function
- Add '&' in font of the parameter that to be called by reference
- More detail will be given in further Lecture (Pointer)

```
void swap(int &a, int &b) {
    int temp;
    temp = a;
    a = b;
    b = temp;
}
int main() {
    int x=1,y=3;
    cout << "x:"<<x <<",y:"<<y<<endl;
    swap(x,y);
    cout << "x:"<<x <<",y:"<<y<<endl;
    return 0;
}
```

Defining classes

```
class class_name {
    public / protected / private:
        attribute1 declaration;
        attribute2 declaration;
        method1 declaration;
        method2 prototype;
};
return_value classname::method2 {
        method body statement;
}
```


Default constructor

- A constructor with no parameters
- Will be called when no argument is given

```
class Circle {
public :
    Circle();
    double getArea();
private :
    int radius;
};
void Circle::Circle() {
    radius = 0;
}
double Circle::getArea() {
    return 3.1415*radius;
}
```

```
int main() {
    Circle circle;
    circle.getArea();
    return 0;
}
```


Default constructors

- A default constructor will be generated by compiler automatically if NO constructor is defined.
- However, if any non-default constructor is defined, calling the default constructor will have compilation error.

```
class Circle {
public :
    Circle(int r);
    double getArea();
private :
    int radius;
};
void Circle::Circle() {
    radius = 0;
}
double Circle::getArea() {
    return 3.1415*radius;
}
```

```
void main(){
    Circle circle;      // illegal
    Circle circle(6); //OK
    circle.getArea();
}
```


Copy constructor

```
class Circle {
public :
    Circle();
    Circle(int r);
    Circle(const Circle& c);
    double getArea();
private :
    int radius;
};
Circle::Circle() {  // default constructor
    radius = 0;
}
Circle::Circle(int r) {// constructor
    radius = r;
}
Circle::Circle(const Circle& c) {// copy constructor
    radius = c.radius;
}
double Circle::getArea() {
    return 3.1415*radius;
}
```

```
void main(){
    Circle circle;      // ok
    Circle circle(6); //OK
    Circle newcircle(circle);
    newcircle.getArea();
}
```


Pointers

- A **pointer** is a variable which stores the address of another variable, i.e. it points to the variable.
- A pointer, like a regular variable, has a type. Its type is determined by the type of the variable it points to.

Variable type	int	float	double	char
Pointer type	int*	float*	double*	char*

Applications of pointer

- Call by Reference
- Fast Array Access
  - Will be covered in later class
- Dynamic Memory Allocation
  - Require additional memory space for storing value.
  - Similar to variable declaration but the variable is stored outside the program.

Pointer examples

```
x = 3;
y = 5;
p1 = &x;
p2 = &y;
y = *p1 - *p2;
p1 = p2;
y = *p1 + *p2;
x = p1 + 1;
```

```
p2 = p1+2;
x = *p2 * *p1;
x = *p2 / *p1;
y = x * *p2;
y += *p1

*p2 += 3;
*p1 *= *p2
```


Call by reference

```
void f (char *c1_ptr) {
    *c1_ptr = 'B';
}
                update
void main() {
    char c1_in_main = ' A'; // c1_in_main = 65
    f(&c1_in_main);
    cout << c1_in_main;   // print 'B'
}
```

c1_ptr points to location 3A8E (that is the variable c1_in_main). *c1_ptr refers to the variable pointed by c1_ptr, i.e. the variable stored at 3A8E

Variable	Variable type	Memory location	Content
c1_in_main	char	3A8E	66
c1_ptr	char pointer	4000	3A8E

c1_ptr = 'B'; // error

Reason: c1_ptr stores a location so it cannot store a char (or the ASCII code of a char)

Call-by-Value

```
int add(int m, int n) {
    int c;
    c = m + n;
    m = 10;
    return c;
}

void main() {
    int a = 3, b = 5, c;
    c = add(a,b);
    cout << a << " " << c << endl;
}
```

Pass value as parameter.

Call-by-Reference

```
int add(int *m, int *n) {
    int c;
    c = *m + *n;
    *m = 10;
    return c;
}

void main() {
    int a = 3, b = 5, c;
    c = add(&a, &b);
    cout << a << " "<< c << endl;
}
```

Pass address as parameter.

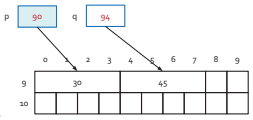
The NULL pointer

- A special value that can be assigned to any type of pointer variable
- A symbolic constant defined in several standard library headers, e.g. **<iostream>**
- When assigned to a pointer variable, that variable points to nothing
- Example

```
int *ptr1 = NULL;
int *ptr2 = 0;
```

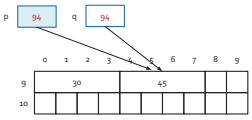

Copy the address

Assignment: p = q;



We copy the content (which is an address) of q to p.

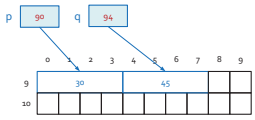
After the assignment, p and q point to the same location in memory.



Therefore, if we change *p, *q will also be changed

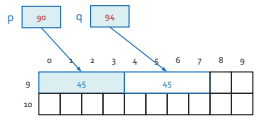
Copy the content

*p = *q;



We copy the value of the variable pointed by q to the variable pointed by p.

After the assignment, p and q still point to different locations in memory.



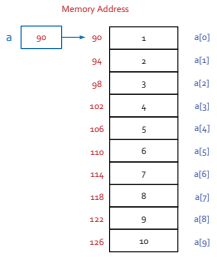
if we change *p, *q will not be changed as p and q points to different location in memory.

Arrays and pointers

Equivalent representation		Remark
num	&num[0]	num is the address of the 0th element of the array
num+i	&(num[i])	Address of the ith element of the array
*num	num[0]	The value of the 0th element of the array
*(num+i)	num[i]	The value of the ith element of the array
(*num)+i	num[0]+i	The value of the 0th element of the array plus i

Example 2: Summing an array

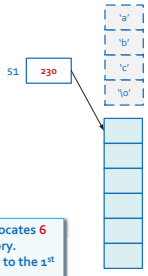
```
#define N 10
int main(){
    int a[N] = {1,2,3,4,5,6,7,8,9,10};
    int i, sum = 0;
    for (i = 0; i < N; ++i)
        sum += *(a + i);
    cout << sum; /*55 is printed*/
    return 0;
}
```



a+1 is the address of a[1]
a+2 is the address of a[2]
...
a+i is the address of a[i]

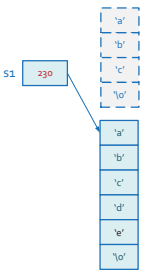
So, *(a+i) means a[i]


```
#include <iostream>
int main () {
    char *s1;
    s1 = new char[4];
    cin >> s1;   /*input "abc"*/
    cout << s1;
    delete s1;
    s1 = new char(6) ; // same as char[6]
    cin >> s1;
    cout << s1;
    delete s1;
    s1 = NULL;
    return 0;
}
```



new dynamically allocates 6 bytes of memory.
new returns a pointer to the 1st byte of the chunk of memory, which is assigned to s1


```
#include <iostream>
int main () {
    char *s1;
    s1 = new char[4];
    cin >> s1;   /*input "abc"*/
    cout << s1;
    delete s1;
    s1 = new char(6) ; // same as char[6]
    cin >> s1;
    cout << s1;
    delete s1;
    s1 = NULL;
    return 0;
}
```



Reading a Line of Characters

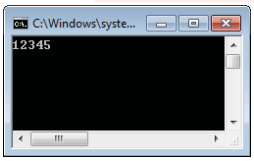
- **cin >> str** will terminate when whitespace characters (space, tab, linefeed, carriage-return, formfeed, vertical-tab and newline characters) is encountered
- Suppose "hello world" is input

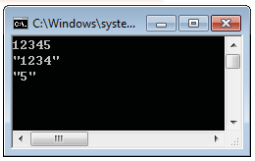
```
char s[20];
cin >> s; //read "hello"
cin >> s; //read "world"
```

- How to read a line of characters?

Example

```
#include <iostream>
using namespace std;
int main() {
    char s[5];
    while (1) {
        cin.getline(s,5);
        cin.clear();
        cout << "\"" << s << "\"" << endl;
    }
    return 0;
}
```





The Null Character '\0'

- The null character,'\0', is used to mark the end of a **cstring**
- '\0' is a single character (although written in two symbols)
- It is used to distinguish a **cstring** variable from an ordinary array of characters (**cstring** variable must contain the null character

H	e	l	l	o	\0				

c	s	2	3	1	1	\0			

Function : count

The size 100 is optional

```
int count_c(char s[100], char c) {
    int occurrence = 0;
    int i;  // counter
    for (i=0; s[i] != '\0'; i++) {
        if (s[i] == c)
            occurrence++;
    }
    return occurrence;
}
```

```
int main() {
    char str[50] = "CityU is a very good university";
    int count = count_c(str, 'o');
    cout << "count = " << count << " \n";
    return 0;
}
```


Example

```
#include <iostream>
using namespace std;
int main() {
    char c1 = 'D', c2 = 'e', c3, c4;
    c3 = c1 + 'a' - 'A';  // convert to lowercase
    c4 = c2 + 'A' - 'a';  // convert to uppercase
    cout << "c3=" << c3 << ", c4=" << c4 << endl;  // c3=d, c4=E
    return 0;
}
```


Common cstring functions in <cstring>

Function	Description	Remarks
strcpy (dest,src)	Copy the content of string **src** to the string **dest**	No error check on the size of **dest** is enough for holding **src**
strcat (dest,src)	Append the content of string **src** onto the end of string **dest**	No error check on the size of **dest** is enough for holding **src**
strcmp(s1,s2)	Lexicographically compare two strings, **s1** and **s2**, character by character.	0: **s1** and **s2** is identical >0: **s1** is greater than **s2** <0: **s1** is less than **s2**
strlen(string)	Returns the number of characters (exclude the null character) contain in the string	

Examples:

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

int main() {
    char s1[50] = "String 1";
    char s2[50] = "cs2363 computer programming";
    cout << "s1:" << s1 << endl;
    cout << "s2:" << s2 << endl;

    strcpy(s1,"cs2311");
    cout << "s1:" << s1 << endl;

    strcat(s1,"computer programming");
    cout << "s1:" << s1 << endl;

    cout << "Compare s1 vs s2:" << strcmp (s1,s2) <<endl;
    cout << "Compare s2 vs s1:" << strcmp (s2,s1) <<endl;
    cout << "Compare s1 vs \"cs2311 computer programming\":"
    << strcmp (s1,"cs2311 computer programming") << endl;
    return 0;
}
```


The program

```
#include <iostream>
using namespace std;
int main() {
    char  input[50];      // define an array input with size 50
    int n;                // length of str
    int i;

    cin >> input;
    n = strlen(input);    // compute string length
    for ( i = n-1; i > 0 ; i--)
        cout << input[i];
    return 0;
}
```

I/O Streams

- Predefined console streams in C++
  - **#include <iostream>**
  - **cin** : input stream physically linked to the keyboard
  - **cout** : output stream physically linked to the screen
- File streams class in C++
  - **#include <fstream>**
  - **ifstream** : stream class for file input
  - **ofstream** : stream class for file output
- To declare an objects of class **ifstream** or **ofstream**, use
  - **ifstream fin;** // fin is the variable name
  - **ofstream fout;** // fout is the variable name

ifstream and ofstream

- To declare an **ifsteam** object
  ifstream fin;
- To open a file for reading
  fin.open("infile.dat");
- To read the file content
  fin >> x; //x is a variable
- To close the file
  fin.close();

- To declare an **ofsteam** object
  ofstream fout;
- To open a file for writing
  fout.open("myfile.dat") ;
- To write something to the file
  fout << x; //x is a variable
- To close the file
  fout.close();
- PS: **fin.open()** and **fout.open()** refer to different functions

Examples

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

int main() {
    ifstream fin;
    ofstream fout;
    int x,y,z;

    fin.open("input.txt");
    fout.open("output.txt");
    fin >> x >> y >> z;
    fout << "The sum is "<< x + y + z;
    fin.close();
    fout.close();
    return 0;
}
```

3 4 7

The sum is 14

Examples: file dump (integer only)

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

int main() {
    ifstream fin;
    int x;
    fin.open("input.txt");
    while (fin >> x) {
        cout << x << " ";
    }
    return 0;
}
```

return 0 if fin has no more data

CS2311 Computer Programming

Consultation

10th – 12th Dec

Time : request by E-Mail

Keep Up

&

Good Luck!