CS2310 Computer Programming

Final Revision

Computer Science, City University of Hong Kong

Semester A 2023-24

Basic syntax

Variable scope

- Global variables are **NOT** recommended
- Scope in user-defined functions

```
int sum(int x, int y)
{
    return x+y;
}

int main() {
    int x, y;
    cin >> x >> y;
    cout << sum(x,y);
    return 0;
}
```

int and char

- Escape sequences

  '\n', '\t', '\#', '\0', '\'', '\"'

- Characters are almost the same as integers

```
char c = 'd';
c++;
cout << c;
```

Output is 'e'

```
char c = 'd';
cout << (char)(c+1);
```

```
char c = 'd';
cout << c+1;
```

Output is 101

C-style string

- Strings are a special kind of arrays (will be covered later)

```
char name[] = "CS2311 CityU";
```

  - Size is optional; variable identifier indicates both an array and a constant pointer; etc.

Conversion between types

```
double z;
z = 1/3;
cout << z;
```

Output is 0

```
double z;
z = 1.0/3;
cout << z;
```

Output is 0.333333

```
char c = 'd';
cout << (char)(c+1);
```

Output is 'e'

Operators

- Increment and decrement operators

```
int x = 0;
x++;
--x;
```

- Efficient assignment operators

```
int x = 4;
x += 1;
x %= 2;
cout << x;
```

Output is 1

Operators

- Logical operators (different from mathematics; commonly used in loops)

```
char x;
cin >> x;
if ('a' <= x <= 'z')
cout << "lowercase" << endl;
```

```
char x;
cin >> x;
if ('a' <= x && x <= 'z')
cout << "lowercase" << endl;
```

Operators

Equality operator (different from the assignment operator!)

```
int x=0;
if (x == 0)
    cout << "false" << endl;
```

Output is false

```
int x=0;
if (x = 0)
    cout << "false" << endl;
```

Output is

cout formatting

- Remember to add the following:

```
#include<iomanip>
```

- Syntax:

```
double x=0.1234567;
cout << fixed << setprecision(2) << x;
```

Output is 0.12

Conditional statements

if… else…

- The if statement can only have one statement in its body
- So it's strongly recommended to always use a compound statement

```
if (mark >= 90) {
    cout << "Excellent!\n";
}
```

dangling else

The else part always matches the NEAREST if

```
if (a==1) {
    if (b==2) {
        cout << “***\n";
    }
    else {
        cout << “###\n”;
    }
}
```

Short-circuit evaluation

- Applies to logical **AND** and **OR** operators
- The left part is always evaluated. The right part may or may not be evaluated
- The key is to remember the truth table for the two operators

Conditional operator

- Usually used as a concise way for expressing simple conditional statements.
- The part before ":" applies when the condition is true

```
int x, y;
cin >> x >> y;
int min_x = (x<y) ? x : y;
cout << min_x;
```

Loops

while

- Basic syntax; always use the compound statement
- do… while: the loop body will be run for at least once

for

- Basic syntax; number of iterations

```
int sum = 0;
for (int i=0; i<10; i++) {
    sum += i;
}
```

- Always a good practice to initialise a variable before use

- Nested for loops, remember to use different index variables

break, continue

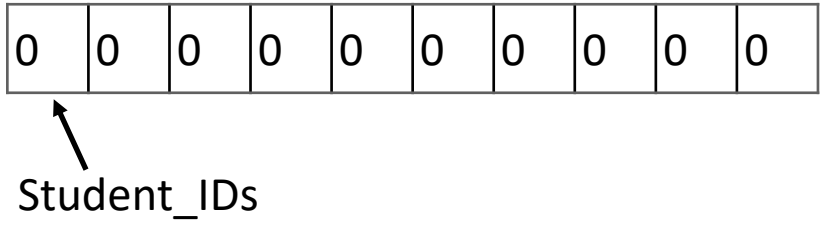
- break causes the control flow to exit from the innermost loop or switch statement
- continue causes the control flow to directly jump to the end of the current iteration, i.e. the start of the next iteration
- break, control exists from the loop; continue, control is still inside the loop, but just skip the rest of the current iteration

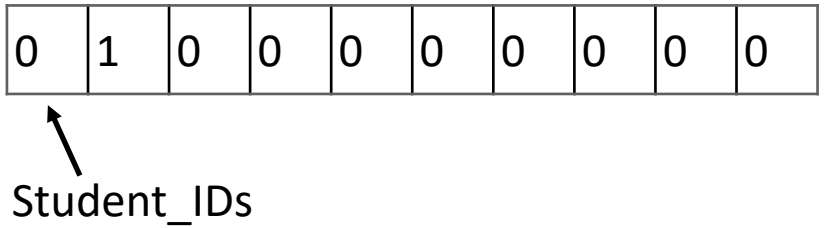
Arrays and Cstring

Declaration and initialization

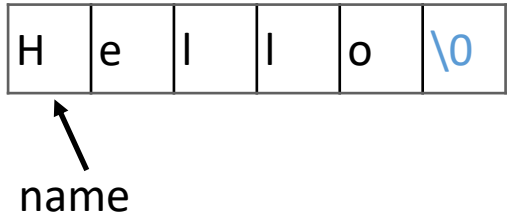
```
int Student_IDs[10];
for(int i=0; i<10; i++)
    Student_IDs[i] = 0;
```

```
int Student_IDs[10] = {0,1};
```






```
char name[6] = "Hello";
char name[] = "Hello";
```



Array operations

- The first element has an index of “0”, not “1”
- Check out-of-bound access
- Know how to use the Bubble sort algorithm

Out-of-bound access

```
int sum(int numbers[], int size) {
    int result = 0;
    for (int i=0; i<size; i++)
        result += numbers[i];
    return result;
}

int main() {
    int numbers[10] = {2,3,5,7,11};
    cout << "Sum is " << sum(numbers, 10);
    return 0;
}
```

Multi-dimensional array

```
// read marks of every student, for every question
int marks[126][9];
int i, j;
for (i=0; i<126; i++) {
    for (j=0; j<9; j++) {
        cin >> marks[i][j];
    }
}
// compute average mark for question 9
int result = 0;
for (i=0; i<126; i++) {
    result += marks[i][8];
}
cout << "Average mark for Q.9 is " << result/126.0;
```

Multi-dimensional array

```
// print each cstring stored in a 2D char array
char strs[2][10]={"hello", "world"};

for (i=0; i<2; i++) {
    cout << strs[i] << endl;
}
```

Reading Cstring

cin.get(), cin.getline()

```
char s[20];
cin >> s;
cin.get()
```

What's the difference?

```
char s[20];
cin.getline(s, 20);
cout << s << endl;
```

hello Hello

hello Hello

End-of-string

Always remember to set '\0' for strings when you are dealing with strings

```
char s1[20] = "Christmas";
char s2[20];
int i;
for (i=0; s1[i] != '\0'; i++) {
    s2[i] = s1[i] + 1;
}
s2[i] = '\0';
cout << s2 << endl;
```

Output: Disjtunbt

Function

Declaration

```
int sum(int numbers[], int size) {
    int result = 0;
    for (int i=0; i<size; i++)
        result += numbers[i];
    return result;
}
```

Variables in a function

```
int sum(int numbers[], int size) {
    int result = 0;
    for (int i=0; i<size; i++)
        result += numbers[i];
    return result;
}

int main() {
    int numbers[10] = {2,3,5,7,11};
    cout << "Sum is " << sum(numbers, 10);
    return 0;
}
```

local to the function only

Parameter passing

call by pointer

call by value

```
int sum(int numbers[], int size) {
    int result = 0;
    for (int i=0; i<size; i++)
        result += numbers[i];
    return result;
}

int main() {
    int numbers[10] = {2,3,5,7,11};
    cout << "Sum is " << sum(numbers, 10);
    return 0;
}
```

Pass by Reference

```
void sum(int numbers[], int size, int &result) {
    for (int i=0; i<size; i++)
        result += numbers[i];
}

int main(){
    int numbers[10] = {2,3,5,7,11};
    int result = 0;
    sum(numbers, 10, result);
    cout << "Sum is " << result;
    return 0;
}
```

Variable reference

Function prototype

```
void sum(int numbers[], int size, int &result);

int main(){
    int numbers[10] = {2,3,5,7,11};
    int result = 0;
    sum(numbers, 10, result);
    cout << "Sum is " << result;
    return 0;
}

void sum(int numbers[], int size, int &result) {
    for (int i=0; i<size; i++)
        result += numbers[i];
}
```

File IO

Synatx

```
#include <fstream>
using namespace std;
void 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();
}
```

3 4 7

The sum is 14

Don’t forget close your file!

Common mistake

```
if(!fin.fail() && !fout.fail()) {
    // while (!fin.eof()){   Wrong!
    while (fin >> x){
        fout << x << " ";
    }
}
```

Warning: Do not mix while with eof().

Pointers

Basics

```
int *p1 = NULL;
int c = 1;
p1 = &c;
cout << *p1 << endl;
cout << p1 << endl;
```

Output:

1
0x7fff5fbff8cc

```
char *p1;
char c = 'a';
p1 = &c;
cout << *p1 << endl;
```

Output:

a

```
char *p1;
char s[] = "Eason Chan";
p1 = s;
cout << *p1 << endl;
cout << p1 << endl;
```

Output:

E
Eason Chan

Copying

`p1 = p2;`	**copy address, pointer assignment**
`*p1 = *p2;`	**copy content**

Pointers and Arrays

```
double x[2] = {1.1,2.2};
double *p;
p = x;
cout << p[1] << endl;
```

Output: **2.2**

Dynamic Memory

1. “new” an array
2. “delete”

```
double *p1;
p1 = new double[2];
p1[0] = 1.1, p1[1] = 2.2;
cout << p1[1] << endl;
delete [] p1;
p1 = NULL;
```

2D Version

```
ifstream in("score.txt");
if (in.fail()) {exit(1);}
int n;
in >> n;
int **p = new int*[n];
for (int i = 0; i < n; i++) {
  p[i] = new int[3];
  for (int j = 0; j < 3; j++)
       in >> p[i][j];
}
in.close();
for (int i = 0; i < n; i++) {
  for (int j = 0; j < 3; j++) {
       if (p[i][j] < 60) {
         for (int k = 0; k < 3; k++)
           cout << p[i][k] << ' ';
         cout << endl;
         break;
       }
  }
}
for (int i=0; i<n; i++) {
     delete [] p[i];
}
delete[] p;
```

Create a 2D dynamic array

Computing

Release the memory

score.txt:

```
4
85 89 64
93 82 94
55 92 59
59 88 70
```

Classes and objects

Declaration and Definition

```
// declaration
class Student
{
private:
    char gender;
    int  id;
public:
    char get_gender();
    void set_gender(char c);
    int  get_id();
    void set_id(int i);
};
```

```
// Function/method definition
int Student::get_id()
{
    return id;
}

void Student::set_id(int i)
{
    id = i;
}
```

Call Class function

```
int main(){
    Student Helen;
    Student& ref2Helen = Helen;
    Student* point2Helen = &Helen;
    Helen.set_id(50001111);
    // Helen.set_gender('F');
    cout << "Helen's ID is ";
    cout << Helen.get_id() << endl;
    cout << ref2Helen.get_id() << endl;
    cout << point2Helen->get_id() << endl;
    return 0;
}
```

Constructor

The default constructor will be synthetically generated, when there is no user-defined constructor. The default constructor just creates the variables and the object.

```
public:
    char get_gender();
    void set_gender(char c);
    int get_id();
    void set_id(int i);
    Student(char c, int i);
    Student();
```

```
Student::Student(char c, int i){
    gender = c;
    id = i;
}
Student::Student(){
    gender = '?';
    id = 0;
}
```

Constructor

```
public:
    char get_gender();
    void set_gender(char c);
    int get_id();
    void set_id(int i);
// This is also a default constructor
    Student(char c='?', int i=0);

Student::Student(char c, int i){
    gender = c;
    id = i;
}
```

```
int main(){
    Student Helen('F', 50001111);
    Student Mike;
    cout << Mike.get_id() << endl;
    Mike = Student('M', 50001113);
    cout << Mike.get_id() << endl;
    return 0;
}
```

Output **0**
50001113

Initializer List

```
private:
    char gender;
    int  id;
public:
    char get_gender();
    void set_gender(char c);
    int get_id();
    void set_id(int i);
// This is also a default constructor
    Student(char c='?', int i=0): gender(c), id(i){};
```

```
int main(){
    Student Helen('F', 50001111);
    Student Mike;
    cout << Mike.get_id() << endl;
    Mike = Student('M', 50001113);
    cout << Mike.get_id() << endl;
    return 0;
}
```

Output

0
50001113

Inheritance and Run-time Polymorphism

```
class CityUPerson {
protected:
    int id;
    char gender;
public:
    CityUPerson(int i=0, char c='?'): id(i), gender(c){};
    virtual ~CityUPerson(){};
    virtual void displayProfile();
};
void CityUPerson::displayProfile(){
    cout << "My id is " << id << endl;
    cout << "My gender is " << gender << endl;
}
```

```
class Student: public CityUPerson {
private:
    int numCourse;
    int year;
public:
// initializer list with arguments passed to Base constructors
    Student(int id=0, char gender='?', int n=0, int y=0):
CityUPerson(id, gender), numCourse(n), year(y) {};
// override the method to display course num & year
    void displayProfile();
};
void Student::displayProfile(){
    cout << "My id is " << id << endl;
    cout << "My gender is " << gender << endl;
    cout << "I have taken " << numCourse << " courses" << endl;
    cout << "I am a Year " << year << " student" << endl;
}
```

Inheritance and Run-time Polymorphism

```
int main(){
    CityUPerson testPerson;          // default constructor
    testPerson.displayProfile();
    Student testStudent;             // default constructor
    testStudent.displayProfile();
    CityUPerson* Helen = new Student(50001111, 'F', 6, 4);
    Helen->displayProfile();         // dynamic binding
    delete Helen;
    return 0;
}
```

TLQ Survey

- Fill in the TLQ learning survey
  - https://onlinesurvey.cityu.edu.hk/tlq/
  - Provide your valuable feedback on CS2311!

Good Luck!