CS2311 Final Revision Part I: Pre-midterm

Dr. Henry Xu

The final exam

- 60% of your grade
- 2-hour, closed-book
- Covers everything we taught
- 8 questions
- Find errors in a program, write code, etc.

Before midterm:

- Basic syntax
- Conditional statements
- Looping statements
- In revision, we will mix different contents if needed

Basic syntax

Variable scope

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

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

int main(){
    int x, y;
    cin >> x >> y;
    cout << summ(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`

Strings (cstring)

- Strings are a special kind of arrays (will be covered in Part 2)

```
char name[] = "Henry Xu";
```

- Size is optional; string identifier is 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) ? y : x;
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

CS2311 Final Revision Part II: Post-midterm

Henry Xu

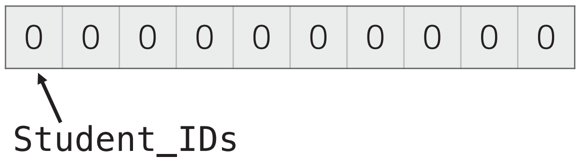
Content

- Arrays and strings
- Functions
- Classes and objects
- Pointers

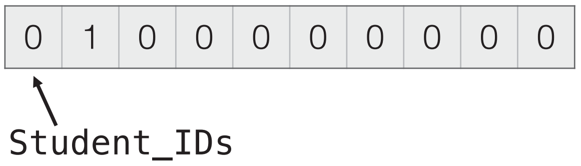
Arrays and strings

Definition and initialization

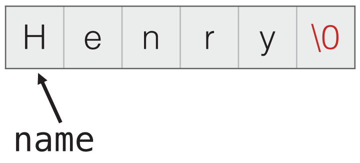
```c
int Student_IDs[10];
for(int i=0; i<10; i++)
    Student_IDs[i] = 0;
```



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



```c
char name[6] = "Henry";
char name[] = "Henry";
```



Common mistakes

- The first element has an index of “0”, not “1”
- Check out-of-bound access
- Try to understand 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 arrays

```
// 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;
```

String input

- cin, cin.getline()

```
char s[20];
cin >> s;
cin >> s;
cout << s;
```

```
hello Henry
Henry
```

```
char s[20];
cin >> s;
cout << s;
```

```
Henrrrrrrrrrrrrrrrry
ERROR
```

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

```
hello Henry
hello Henry
```

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**

Functions

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 reference   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;
}
```

Call by reference

Variable 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;
}
```

Prototypes

```
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];
}
```

Classes and objects

Declaration

```
class student
{
private:
    char sex;
    int id;
public:
    char get_sex();
    void set_sex(char c);
    int get_id();
    void set_id(int i);
};
```

Method definition

```
int student::get_id()
{
    return id;
}

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

```
int main(){
    student Helen;
    Helen.set_id(50001111);
    Helen.set_sex('F');
    cout << "Helen's ID is " << Helen.get_id() << endl;
    return 0;
}
```

Objects

object assignment =

```
int main(){
    student Helen, best_student;
    Helen.set_id(50001111);
    Helen.set_sex(‘F');

    best_student = Helen;

    cout << "The best student's ID is "
    << best_student.get_id() << endl;
    return 0;
}
```

best_student and Helen both point to the same object in memory (*copy by reference*)

Constructors

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

```
public:
    char get_sex();
    void set_sex(char c);
    int get_id();
    void set_id(int i);
    student(char c, int i);
    student();
```

```
student::student(char c, int i)
{
    sex = c;
    id = i;
}

student::student()
{
    sex = '?';
    id = 0;
}
```

Constructors

```
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
```

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
```

cout treats char pointer differently, as a string

Copying

```
p1 = p2;
*p1 = *p2;
```

copy address
copy content

Pointers and arrays

```
double x[2] = {1.1,2.2};
double *p1;
p1 = x;
cout << p1[1] << endl;
```

Output: `2.2`

```
double x[2][2] = {1.1,2.2,3.3,4.4};
double *p1;
p1 = x[1];
cout << p1[1] << endl;
```

Output: `4.4`

Dynamic memory allocation

1. “new” an array
2. “delete”
3. make the pointer point to NULL

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

Programming style

Good practice

- Proper variable naming
- No global variable
- Indentation
- Comments
- Use functions whenever possible
- Initialize a variable before use
- Initialize a pointer to NULL; make it NULL after free

Good luck!