Week 3: Strings and Functions

Ling Ding
Email: lingding@cs.ucla.edu

Slides modified from Muhao Chen, with permission

Outline

- Char in C++
- String in C++
- Function
- Break, continue, return

Outline

- *Char in C++*
- String in C++
- Function
- Break, continue, return

Char in C++

- Character type **`char`** is encoded using a schema of 1 byte integers (i.e. ASCII)
- Range (0~255)
- ASCII is the dominant encoding scheme
  - Examples
    - `' '` encoded as 32
    - `'A'` encoded as 65
    - `'a'` encoded as 97
    - `'+'` encoded as 43
    - `'Z'` encoded as 90
    - `'z'` encoded as 122

Char in C++

- Arithmetic and relational operations are defined for characters types
  - **'a' < 'b'** is true
  - **'4' > '3'** is true
  - **'6' <= '2' is false**
  - **‘F’ – 5 is ‘A’**
  - **‘x' + (‘A’ – ‘a’) is ‘X’**
  - **‘Y’ – (‘Z’ – ‘z’) is ‘y’**

Lower case letter is actually greater than its upper case (-32)

Char in C++

- Explicit (literal) characters within single quotes
  - `'a','D','*'`
- Special characters - delineated by a backslash \
  - Two character sequences (escape codes)
  - Some important special escape codes
    - `\t` denotes a tab
    - `\\` denotes a backslash
    - `\"` denotes a double quote
    - `\n` denotes a new line
    - `\'` denotes a single quote
    - `\0` 0, end of string
  - `'\t'` is the explicit tab character, `'\n'` is the explicit new line character, and so on

Char in C++

- #include<cctype> provides several functions to process char, e.g.:
  - isdigit(char c): Is c a digit?
  - islower(char c): Is c lower case?
  - isupper(char c): Is c upper case?
  - isalpha(char c): Is c alphabetic?
    - Yes->return true, No->return false
  - tolower(char c): Convert c to lower case
  - toupper(char c): Convert c to upper case

Char in C++

- Example

```
// This program demonstrates some of the character testing
// functions.
#include <iostream>
#include <cctype>

void main(void)
{
   char input;
   cout << "Enter any character: ";
   cin.get(input);
   cout << "The character you entered is: " << input << endl;
   cout << "Its ASCII code is: " << int(input) << endl;
```

Char in C++

- Example (cnt.)

```
    if (isalpha(input))
        cout << "That's an alphabetic character.\n";
    if (isdigit(input))
        cout << "That's a numeric digit.\n";
    if (islower(input))
        cout << "The letter you entered is lowercase.\n";
    if (isupper(input))
        cout << "The letter you entered is uppercase.\n";
    if (isspace(input))
        cout << "That's a whitespace character.\n";
}
```

Char in C++

- Example (cnt.)
  - Input: 1
  - Input: A

```
Enter any character: 1
The character you entered is: 1
Its ASCII code is: 49
That's a numeric digit.
Press any key to continue . . .
```

```
Enter any character: A
The character you entered is: A
Its ASCII code is: 65
That's an alphabetic character.
The letter you entered is uppercase.
Press any key to continue . . .
```

Outline

- Char in C++
- *String in C++*
- Function
- Break, continue, return

String in C++

- String is a class in C++;
  - Class:
    - We will learn Class in detail in later classes.
    - Similar to Type, but more powerful than Type, e.g. it can define its own functions and attributes.
    - A C-string is a sequence of characters stored in consecutive memory locations, terminated by a null('\0') character.
  - To use string, we need to add
    - #include<string>

String in C++

- String consists of characters (i.e. internally it's a dynamic array of char)
- For example: string s = "ab cd";
  - s consists of 5 characters: 'a', 'b', ' ', 'c', 'd';
  - We can use s.size() to get the number of characters in s, i.e. 5.
  - We can use s[i] to access the **(i+1)-**th character in s, e.g. s[1] = 'b'. (i = 0 .. s.size()-1)
    - Type of s[i] is char
  - **Using s[i], i greater than s.size()-1 is an undefined behavior**

String in C++

```
cin >> s;
…
for (int k=0; k<s.size(); k++) {
    if(s[k] == ‘H’) {
        if(s[k+1] == ‘E’)
            countHE++;
    }
}
```

//SHELLFISH

Outline

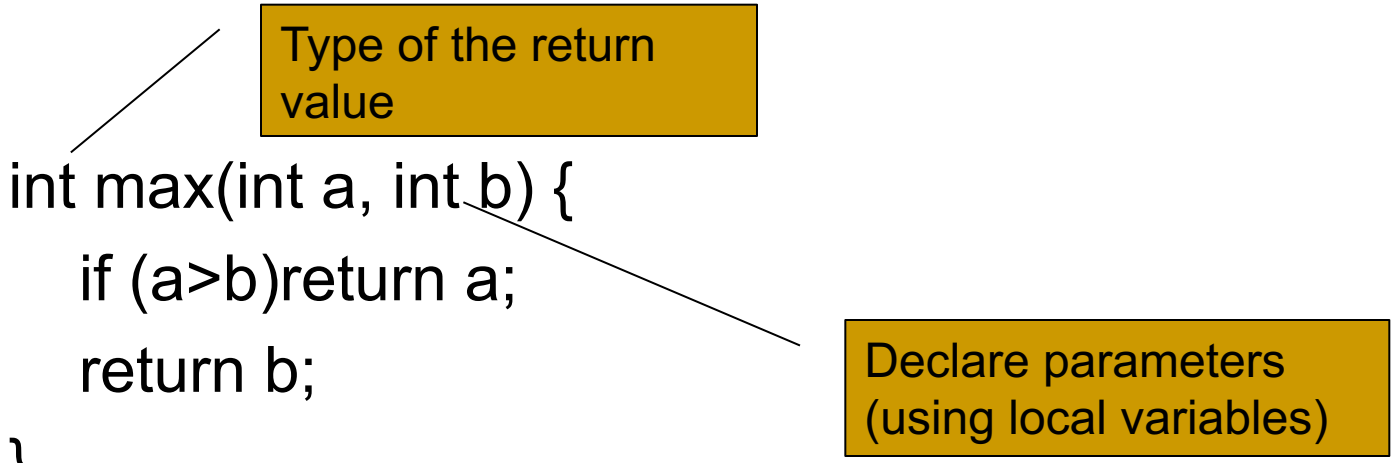
- Char in C++
- String in C++
- *Function*
- Break, continue, return

Function

A batch of code. Input some parameters to the function, run a procedure, return a result.

Y=f(X)

A Simple Function




```
int max(int a, int b) {
    if (a>b)return a;
    return b;
}


int main(){
    int a;
    //call the function
    a=max(4,6); //a is 6
}
```

Another function

- //trim(). Remove all ‘ ‘ from the beginning and the end of a string.

```
string trim(string str) {
    string result=“”;
    int i,j;
    for (i=0; str[i]==‘ ‘ && i<str.size(); ++i);
    for (j=str.size() - 1; j>=i && str[j]==‘ ‘; --j);
    for (int k=i; k<=j; ++k)result+=str[k];
    return result;
}
int main(){
    string s = “    Galneryus is a great band.     ”
    s = trim(s);
    cout<<s<<endl;  // “Galneryus is a great band.”
}
```

return str.substr(i, j-i);

Void

- A function whose return type is void: no return value. Also known as a procedure or subprogram in some other languages (e.g. pascal).

```
void printFactorial(int n) {
    int prod = 1;
    for (int i = 2; i <= n; i++)
            prod *= i;
    cout << "The factorial of " << n << " is " << prod << endl;
}
```

Example

```
void trim(string &str) {
    int i, j;
    for (i=0; str[i]==‘ ‘ && i<str.size(); ++i);
    for (j=str.size()-1; str[j]==‘ ‘ && j>=i; --j);
    str = str.substr(i, j-i);
    return;
}
```

Void

- void f(…);
- int a; a=f(); X
- No return value!

Main()

- main() is also a function. The operating system calls main() to start the program.
- return 0 of main()
  - In some environment (e.g., remote procedure call in a distributed system), the system need to know if a program ends successfully by returning a 0.

Forbidden in functions

```
void function(int a, int b) {
    int a;
    double b;
    …
}
```

1. Define local variables using the same name of parameters. (Compile error) X

Forbidden in functions

```
bool non_negative(int a) {
        if (a>0)return true;
        else if(a<0)return false;
}
```

2. Certain condition causes no return of a function. (undefined behavior; compile error in some IDE) X

Forbidden in functions

```
int function(int a, int b){
    double c = 1.0 * a / b;
    return c;
}
```

3. Inconsistent type of return value. X

Forbidden in functions

```
int function1() {
    int function2() {
            …
    }
    …
}
```

4. Nested definition of functions. X

Where to place the defined function

- Before the caller

```
int func(int a, int b){
    …
}

int main() {
    …
    c=func(4,6);
    …
}
```

- After the caller
  - Require signature

```
int func(int, int);

int main() {
    …
    c=func(4,6);
    …
}

int func(int a, int b){
    …
}
```

Can we define multiple functions with the same name

- Only if they have different combinations and/or types of parameters. (**Overloading**)

- int func(int a)
- int func(int a, int b)
- int func(int a, float b)
- int func(float c, int d)

Example

```
//func1
int max(int a, int b) {
    if(a>b)return a;
   return b;
}
```

```
//func2
float max(float a, float b) {
    if(a>b)return a;
   return b;
}
```

```
//func3
int max(int a, int b, int c) {
    //call func1
    return max(a, max(b, c));
}
```

We can always call a function in another function.

```
int main(){
    //call func1
    cout<<max(2, 3)<<endl;
    //call func2
    cout<<max(3.7, 4.0)<<endl;
    //call func3
    cout<<max(3,4,5)<<endl;
}
```

Can we define multiple functions with the same name

double func(int a, int b)

int func(int a, int b)

X

Signature of a function: name, #parameters, type of parameters. (Do not incl. type of return)

Formal Parameters & Reference Parameters

- **Formal parameters**: we want to pass some value to a function, but won't change the values of the variables we use to pass the value

```
int max(int a, int b) {
    if(a>b)return a;
   return b;
}

…
int x=4, y=6;
cout<<max(x,y);
```

Use the vals of x,y. Do not change their vals. It's OK.

```
void swap(int a, int b) {
    int tmp=a;
    a=b;
    c=a;
}
…
int x=4, y=6;
swap(x,y);
```

Change the vals of x,y? It won't work! X is still 4, y is still 6.

```
void mod(int a, intb) {
    a%=b;
}
…
int x=40, y=6;
mod(x,y);
```

Again, it won't work! X is still 40.

Reference Parameters (actual parameters)

- **Reference Parameters**: we want to both read and write the passed parameters.

```
void swap(int a, int b) {
    int tmp=a;
    a=b;
    c=a;
}
…
int x=4, y=6;
swap(x,y);
```

Changes to

```
void swap(int &a, int &b) {
    int tmp=a;
    a=b;
    c=a;
}
…
int x=4, y=6;
swap(x,y);
```

x is 6. y is 4

```
void mod(int a, intb) {
    a%=b;
}
…
int x=40, y=6;
mod(x,y);
```

Changes to

```
void mod(int &a, int b) {
    a%=b;
}
…
int x=40, y=6;
mod(x,y);
```

x is 40%6 -> 4

Formal Parameters & Reference Parameters

- An easy way to remember when to use formal / reference parameters.

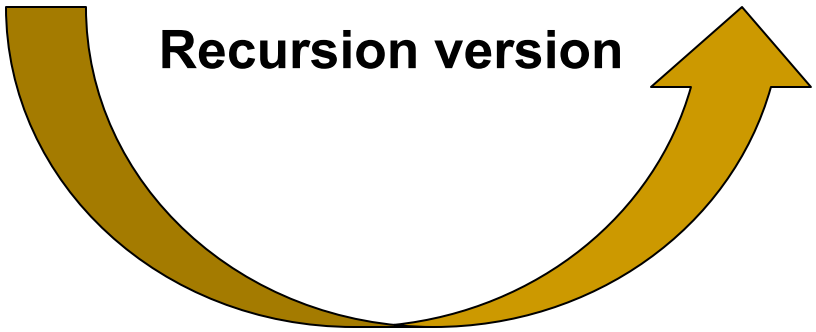
Type	Representation	Functionality
Formal	a	**Read-only**
Reference (actual)	**&a**	**Read-or-write**

Recursion

- A function that calls itself.

```
int factorial(unsigned int n) {
    result = 1;
    for (int i=2;i<=n;++i)result*=i;
    return result;
}
```

```
int factorial(unsigned int n) {
    if (n<=1) return 1;
    else return n * factorial(n-1);
}
```



Recursion

- More complex problems to solve with recursion:
  - Eight queen problem.
  - Hanoi tower.
  - Transitive closure.
  - Etc…

Outline

- Char in C++
- String in C++
- Function
- *Break, continue, return*

Return; Break; Continue

- Return: terminate a function.
- Break: terminate a level of loop.
- Continue: terminate a cycle of loop.

Return; Break; Continue

```
void main () {
    string s= “Nissan GTR”;
    for (int i=0;i<s.size();++i) {
        if(islower(s[i])) return;
        cout<s[i];
    }
    cout<<“ Nismo”;
}
```

```
void main () {
    string s= “Nissan GTR”;
    for (int i=0;i<s.size();++i) {
        if(islower(s[i])) break;
        cout<s[i];
    }
    cout<<“ Nismo”;
}
```

```
void main () {
    string s= “Nissan GTR”;
    for (int i=0;i<s.size();++i) {
      if(islower(s[i]))continue;
      cout<s[i];
     }
    cout<<“ Nismo”;
}
```

N	N Nismo	N GTR Nismo

Thank you!