CS100 Introduction to Programming

Lecture 13 C++ Warm Up

Outline

Basic IO

Namespace std

String

Header

Range-based for-loops

Basic IO

In C

```c
#include <stdio.h>

int main() {

  printf("Hello world\n");
  return 0;
}
```

In C++

```cpp
#include <iostream>

int main() {
  std::cout << "Hello world" << std::endl;
  return 0;
}
```

In C

```c
#include <stdio.h>

int main() {
  int a, b;
  scanf("%d %d", a, b);
  printf("a+b= %d\n", a + b);
  return 0;
}
```

In C++

```cpp
#include <iostream>

int main() {
  int a, b;
  std::cin >> a >> b;
  std::cout <<"a+b= "<< a + b << std::endl;
  return 0;
}
```

Input char array

```
#include <iostream>

int main() {
  char line[10];
  std::cin >> line; //abc def
  std::cout <<"Your input is: "<< line<< std::endl;
  return 0;
}
```

Namespace std

- `std::cin`, `std::cout`, names from the standard library
- To avoid name collision: `namespace std`.

```
#include <iostream>

using std::cin;

int main() {
  int a, b;
  cin >> a >> b; // std::cin -> cin
  std::cout <<"a+b= "<< a + b << std::endl;
  return 0;
}
```

- `using namespace std;` introduces every name in `std`.

```
#include <iostream>

using namespace std;

int main() {
  int a, b;
  cin >> a >> b;
  cout <<"a+b= "<< a + b << endl;
  return 0;
}
```

String

In C we used `char*` to represent a string.
The C++ standard library provides a common implementation of a string class abstraction named string

```
#include <string>
```

`std::string`

In C

```c
#include <stdio.h>
int main()
{
        // create string ‘str’ = “Hello world!”
        const char *str = “Hello World!”;

        // print sring
        printf(“%s\n”, str);
}
```

In C++

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

int main()
{
        // create string ‘str’ = “Hello world!”
        std::string str = “Hello World!”;

        // print sring
        std::cout<< str <<std::endl;
}
```

Comparison

```c
#include <stdio.h>
int main()
{
        // create string ‘str’ = “Hello world!”
        const char *str = “Hello World!”;

        // print sring
        printf(“%s\n”, str);
}
```

```cpp
#include <iostream>
#include <string>
int main()
{
        // create string ‘str’ = “Hello world!”
        std::string str = “Hello World!”;

        // print sring
        std::cout<< str <<std::endl;
}
```

Create a string

```
string str = "Hello world";
// string str("Hello world"); equivalent
cout << str << endl;

string s1(7, 'a');
cout << s1 << endl;

string s2 = s1; // a copy
cout << s2 << endl;
```

Length of a string

```
string str = "Hello world";
cout << str.size() << endl;
```

Not `strlen` Not `sizeof`.

Not null-terminated.

Check whether a string is empty:

```
if (str.empty()) {
  // ...
}
```

String concatenation (`+`, `+=`)

Concatenating one string to another is done by the ‘+’ operator

Operator-overloading will be seen later

```
string str1 = "Here ";
string str2 = "comes the sun";
string concat_str = str1 + str2;
```

`str1 = str1 + str2;` VS `str1 += str2;`

String comparison

To check if two strings are equal use the '==' relational and equality operators (lexicographical order)

```
string str1 = "Here ";
string str2 = "comes the sun";
if ( str1 == str2 )
      /* do something */
else
  /* do something else */
```

String assignment

use `=` directly

semantic: copy (We don't need other functions like `strcpy`)

different from C-style strings:

```
const char *s1 = /* ... */;
const char *s2 = NULL;
s2 = s1; // points to the same string
```

IO of `std::string`

```
string str;
cin >> str; // Discard leading whitespaces, read until whitespace

string line;
getline(cin, line); // Read from the current position to the first newline ('\n')
```

Conversion between `std::string` and integers

`std::to_string`

```
int ival = 42;
double dval = 3.14;
std::string s = std::to_string(ival) + std::to_string(dval);
// s == 423.14
```

`std::stoi, std::stol` ...

https://en.cppreference.com/w/cpp/string/basic_string#Numeric_conversions

We don't use `strtol` or `atoi`: They are for C-style strings.

Access by subscript

Output uppercase letters:

```
for (std::size_t i = 0; i != s.size(); ++i)
  if (std::isupper(s[i]))
    std::cout << s[i] <<" ";
std::cout << std::endl;
```

Header

Header: `<ctype.h>` -> `<cctype>`

For header files inherited from the C library, we recommend using `<cname>` instead of `name.h`.

Names in `<cname>` are also in `namespace std`.

`<stdio.h>` -> `<cstdio>`, `printf` -> `std::printf`

Range-based for-loops

Range-based for-loops

Output uppercase letters:

```
for (std::size_t i = 0; i != s.size(); ++i)
  if (std::isupper(s[i]))
    std::cout << s[i];
std::cout << std::endl;
```

Better (modern) way:

```
for (char c : s)
  if (std::isupper(c))
    std::cout << c;
std::cout << std::endl;
```