Ve 280

RC3

Hou Yuechao

Outline

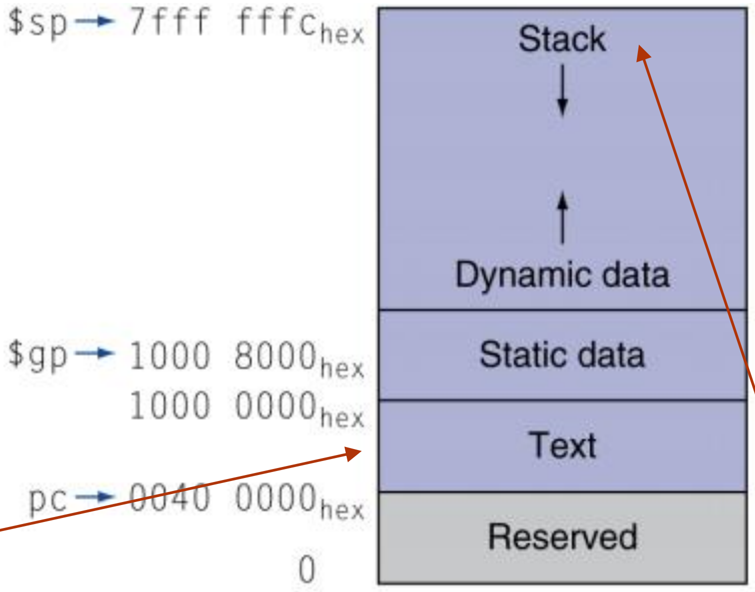
- Function Call Mechanism (quick review)
- Passing Arguments to Program
- I/O Streams
  - File Stream
  - String Stream
- Testing
  - File Stream
  - String Stream
- Exception

Outline

- Function Call Mechanism (quick review)
- Passing Arguments to Program
- Categorizing Data: enum
- I/O Streams
  - File Stream
  - String Stream
- Testing
  - File Stream
  - String Stream
- Exception

Call Stacks(Optional)

```
void fun1(int, int);
void fun2(float);
int main(){
    int x=1;y=2;
    fun1(x, y);
    return 0;
}
void fun1(int a,int b){
    float x=3;
    int y=fun2(x);
}
int fun2(float y) {
    int x=1;
    return 2;
}
```



x	1	main
y	2	
	0	
	return address (old)	
a	1	fun1
b	2	
x	3	
y	0	
	return address (old)	
y	3	fun2
x	1	
	return address (new)	

Call Stacks(Optional)

Dynamic Stack

Static Stack
(static)

Ask for a part of
memory

computer

programmer

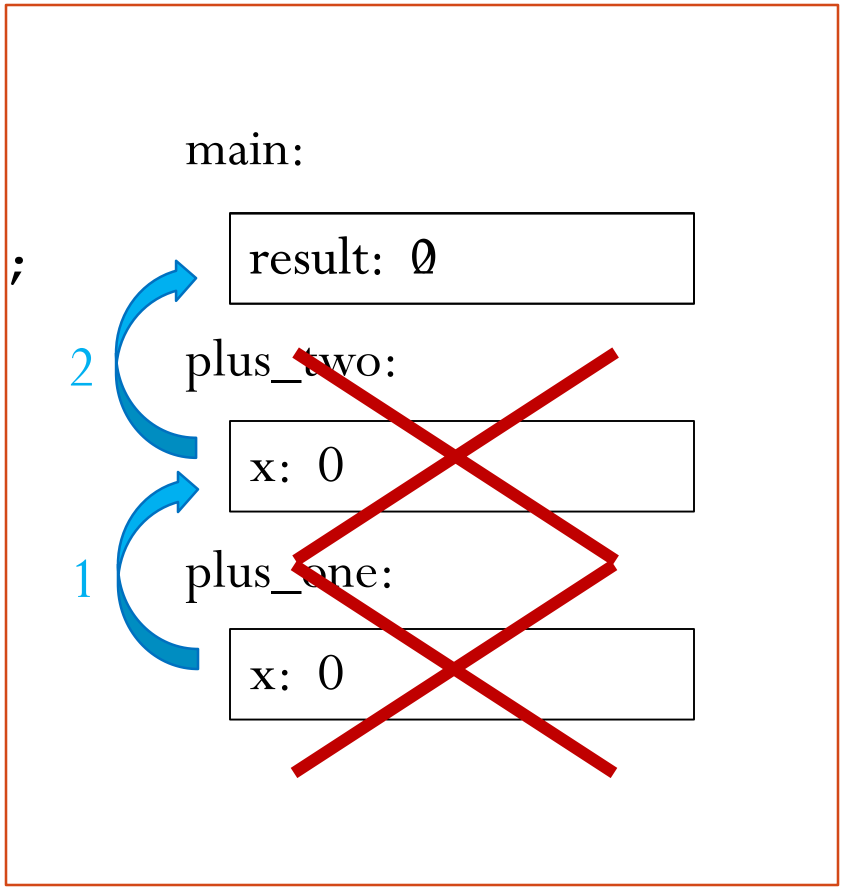
New (remember to delete)

Call Stacks

Example

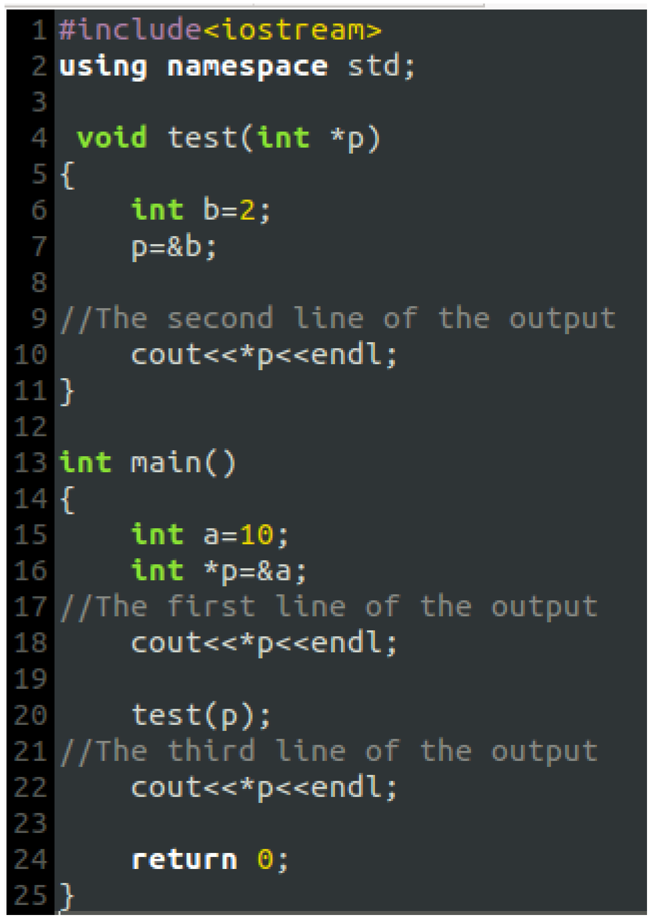
```
int plus_one(int x) {
  return (x+1);
}

int plus_two(int x) {
  return (1 + plus_one(x));
}

int main() {
  int result = 0;

  result = plus_two(0);
  cout << result;
  return 0;
}
```

Dynamic Stack



Call Stacks

Example



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

 void test(int *p)
{
    int b=2;
    p=&b;

//The second line of the output
    cout<<*p<<endl;
}

int main()
{
    int a=10;
    int *p=&a;
//The first line of the output
    cout<<*p<<endl;

    test(p);
//The third line of the output
    cout<<*p<<endl;

    return 0;
}
```

The program outputs three lines
Is there difference between these three lines?

Outline

- Function Call Mechanism
- Passing Arguments to Program
- Categorizing Data: enum
- I/O Streams
  - File Stream
  - String Stream
- Testing
  - File Stream
  - String Stream
- Exception

Passing Arguments to Program

- So far, we have considered programs that take no arguments
  - You run your program like: ./p1
- Arguments are passed to the program through main() function.
- We need to change the argument list of main():
  - Old: `int main()`
  - New: `int main(int argc, char *argv[])`
- `int argc`: the number of strings in the array
  - E.g., `./p1 a1 a2`: `argc = 3`

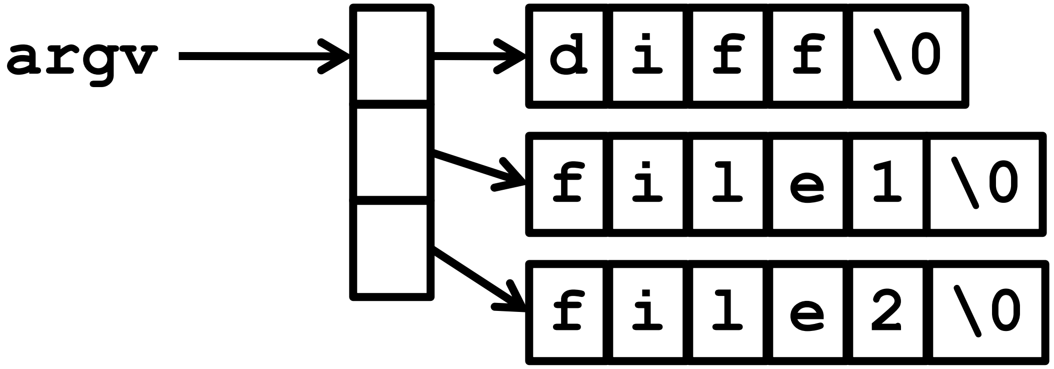
Passing Arguments to Program

argv

```
diff file1 file2
```

```
char *argv[]
```

- Pictorially, this would look like the following in memory:



Note: **argv[0]** is the name of the program being executed.

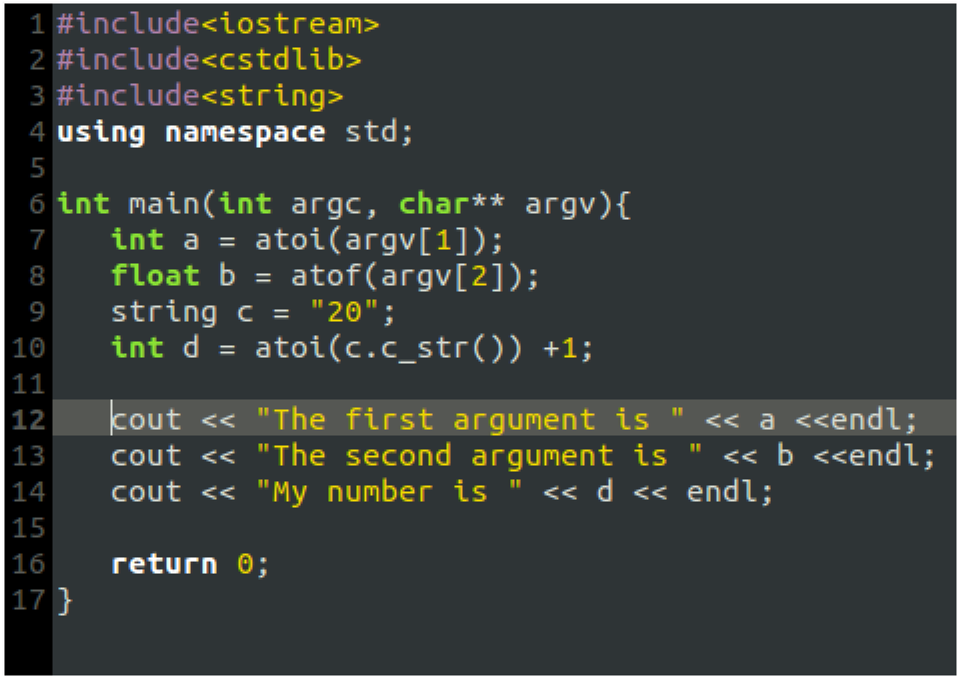
Passing Arguments to Program

Example

Sting and C-string are different types in C++

C-string (char*) to integer: `int atoi(const char *s);`

C-string (char*) to float: `float atof(const char *s);`



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

int main(int argc, char** argv){
   int a = atoi(argv[1]);
   float b = atof(argv[2]);
   string c = "20";
   int d = atoi(c.c_str()) +1;

   cout << "The first argument is " << a <<endl;
   cout << "The second argument is " << b <<endl;
   cout << "My number is " << d << endl;

   return 0;
}
```

Outline

- Function Call Mechanism
- Passing Arguments to Program
- Categorizing Data: `enum`
- I/O Streams
  - File Stream
  - String Stream
- Testing
  - File Stream
  - String Stream
- Exception

Categorizing Data

Introducing enums

- For example, there are four different suits in cards:
  - Clubs ♣
  - Diamonds ♦
  - Hearts ♥
  - Spades ♠
- You can define an enumeration type as follows:

```
enum Suit_t {CLUBS, DIAMONDS, HEARTS, SPADES};
```

- To define variables of this type you say:

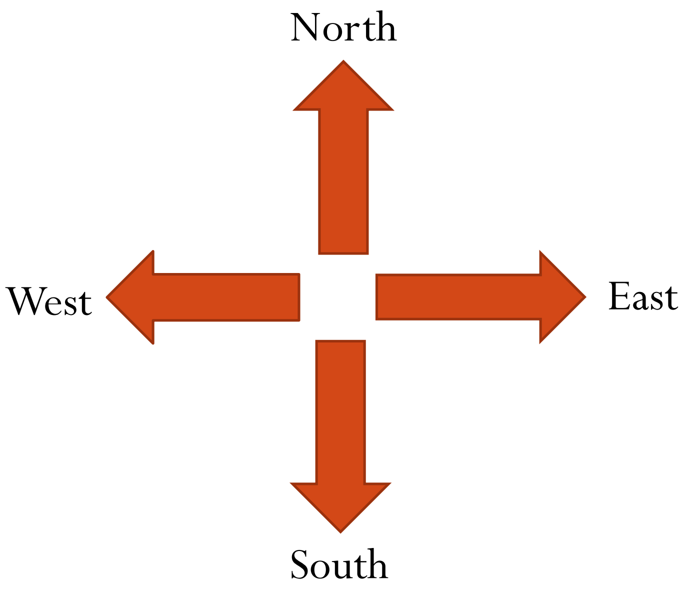
```
enum Suit_t suit;
```

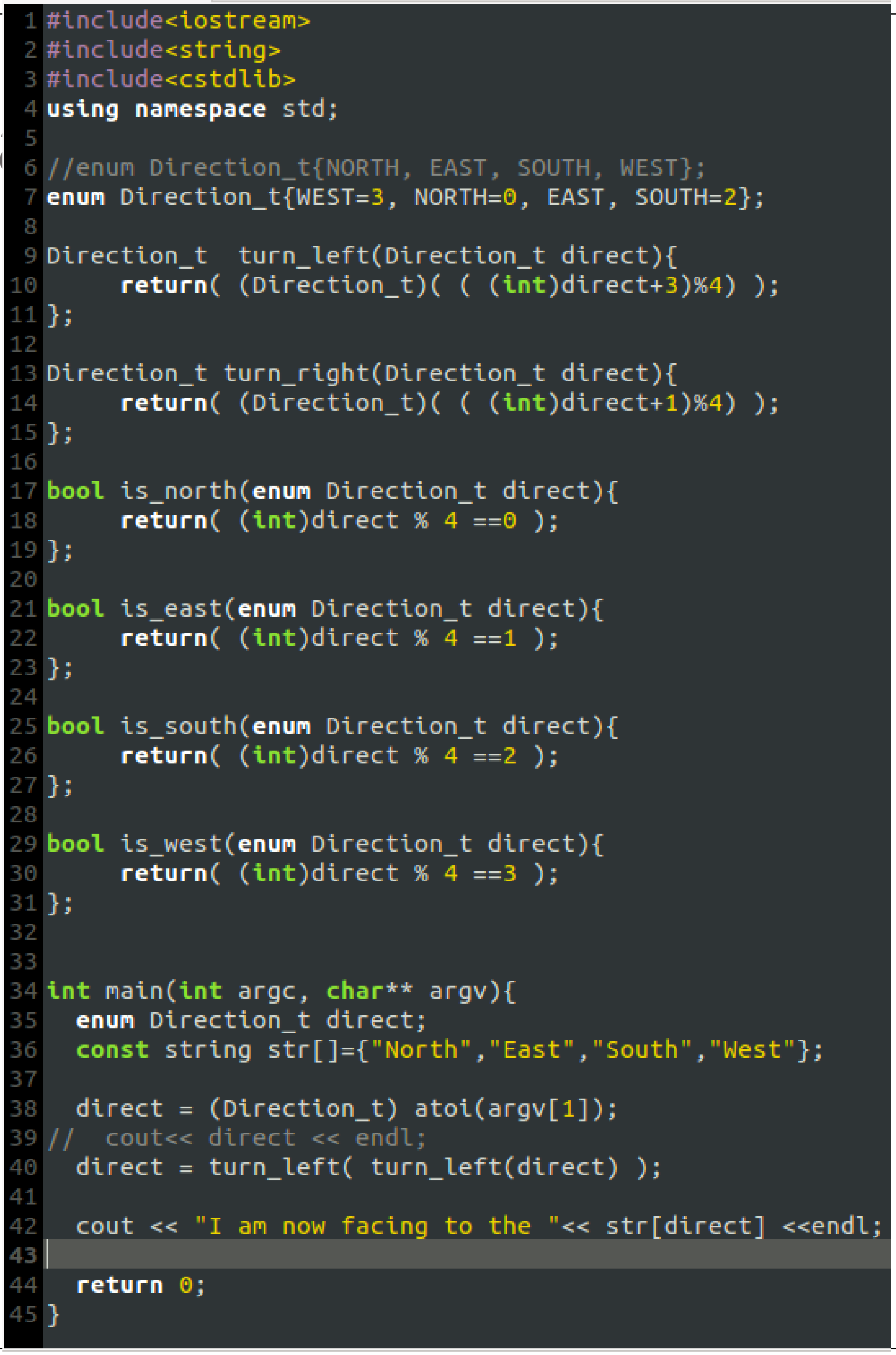
- You can initialize them as:

```
enum Suit_t suit = DIAMONDS;
```

Categorizing D

Introducing enums





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

//enum Direction_t{NORTH, EAST, SOUTH, WEST};
enum Direction_t{WEST=3, NORTH=0, EAST, SOUTH=2};

Direction_t  turn_left(Direction_t direct){
      return( (Direction_t)( ( (int)direct+3)%4) );
};

Direction_t turn_right(Direction_t direct){
      return( (Direction_t)( ( (int)direct+1)%4) );
};

bool is_north(enum Direction_t direct){
      return( (int)direct % 4 ==0 );
};

bool is_east(enum Direction_t direct){
      return( (int)direct % 4 ==1 );
};

bool is_south(enum Direction_t direct){
      return( (int)direct % 4 ==2 );
};

bool is_west(enum Direction_t direct){
      return( (int)direct % 4 ==3 );
};


int main(int argc, char** argv){
  enum Direction_t direct;
  const string str[]={"North","East","South","West"};

  direct = (Direction_t) atoi(argv[1]);
//  cout<< direct << endl;
  direct = turn_left( turn_left(direct) );

  cout << "I am now facing to the "<< str[direct] <<endl;

  return 0;
}
```

Outline

- Function Call Mechanism
- Passing Arguments to Program
- Categorizing Data: enum
- I/O Streams
  - File Stream
  - String Stream
- Testing
  - Specific test cases
  - Incremental testing
- Exception

Input/Output

Streams

- A popular model for how input and output is done in computer systems is centered around the notion of a **stream**.

```
    cin >> a;
```

- Output to screen.

```
    cout << foo << " " << bar << endl;
```

- You can also use the Linux I/O **redirection**:

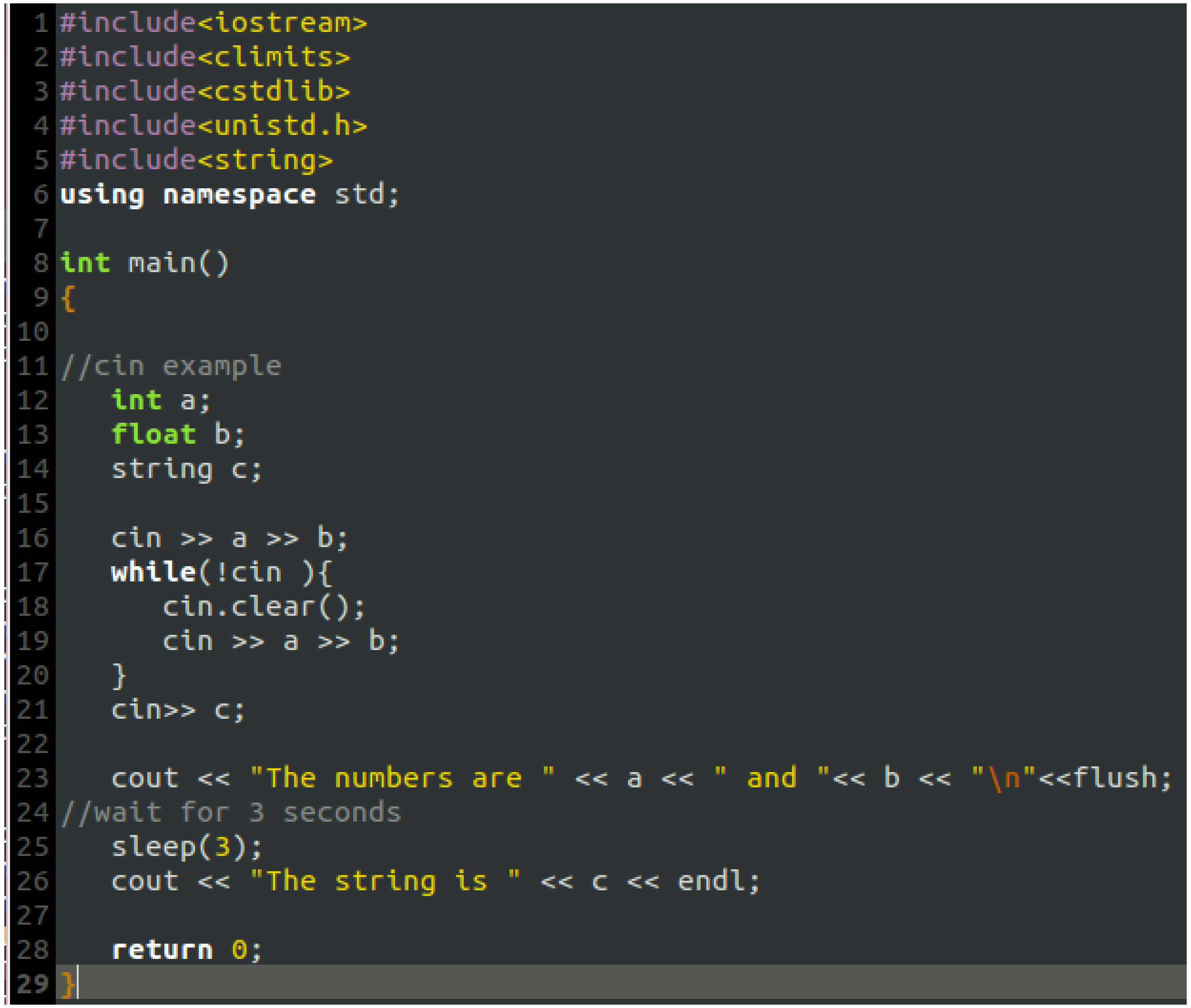
```
    $ ./hello > output.txt
     $ ./hello < foo
```

Buffering

- I/O in C++ is **buffered**.
- The content in the buffer is written to the output only when specific actions are taken.
  - `cout << “ok” << flush;`
  - `Cout << “ok” << endl;`
- Once the buffer content is written to the output, the buffer is **cleaned**
- In contrast, output sent to `cerr` is not buffered

Buffering

example



```
#include<iostream>
#include<climits>
#include<cstdlib>
#include<unistd.h>
#include<string>
using namespace std;

int main()
{

//cin example
   int a;
   float b;
   string c;

   cin >> a >> b;
   while(!cin ){
      cin.clear();
      cin >> a >> b;
   }
   cin>> c;

   cout << "The numbers are " << a << " and "<< b << "\n"<<flush;
//wait for 3 seconds
   sleep(3);
   cout << "The string is " << c << endl;

   return 0;
}
```

getline()

- If you need to read strings including whitespace (**blanks**, **tabs**, or **newlines**), use the `getline()` function:

```
    cin >> foo >> bar;
    getline(cin, baz);
```

Assume inputs is:
`4.2 3.14 four score\n`

- `getline()` reads all characters **up to but not including** the next newline and puts them into the **string variable**, and then **discards the newline**
- But `baz` is " `four score`"; it keeps the leading space

get()

- The `get()` function reads **a single** character, whitespace or newlines:

```
char ch;
cin.get(ch); // Extracts a character
//from cin stream and stores it in ch
```

- So, we can accomplish what we'd hoped to accomplish by:

```
cin >> foo >> bar;
cin.get(ch);
getline(cin, baz);
```

Assume inputs is:

```
42 3.14 four score\n
```

- This makes `baz` “`four score`”.

The three methods have such different syntax.
However, the three methods can be freely intermixed.

Using File Streams

- `#Include <fstream>`
```
    ifstream iFile;
    ofstream oFile;
```
- Connecting a stream to a file is opening the file for the stream
```
  iFile.open("myText.txt");
```

Must be a C-style string, cannot be C++ string!
use c_str() to convert C++ string into C string

To Check whether we can not open the file:
```
iFile.open("a.txt");
if(!iFile) {
    cerr << "Cannot open a.txt\n";
    return -1;
}
```

String Stream

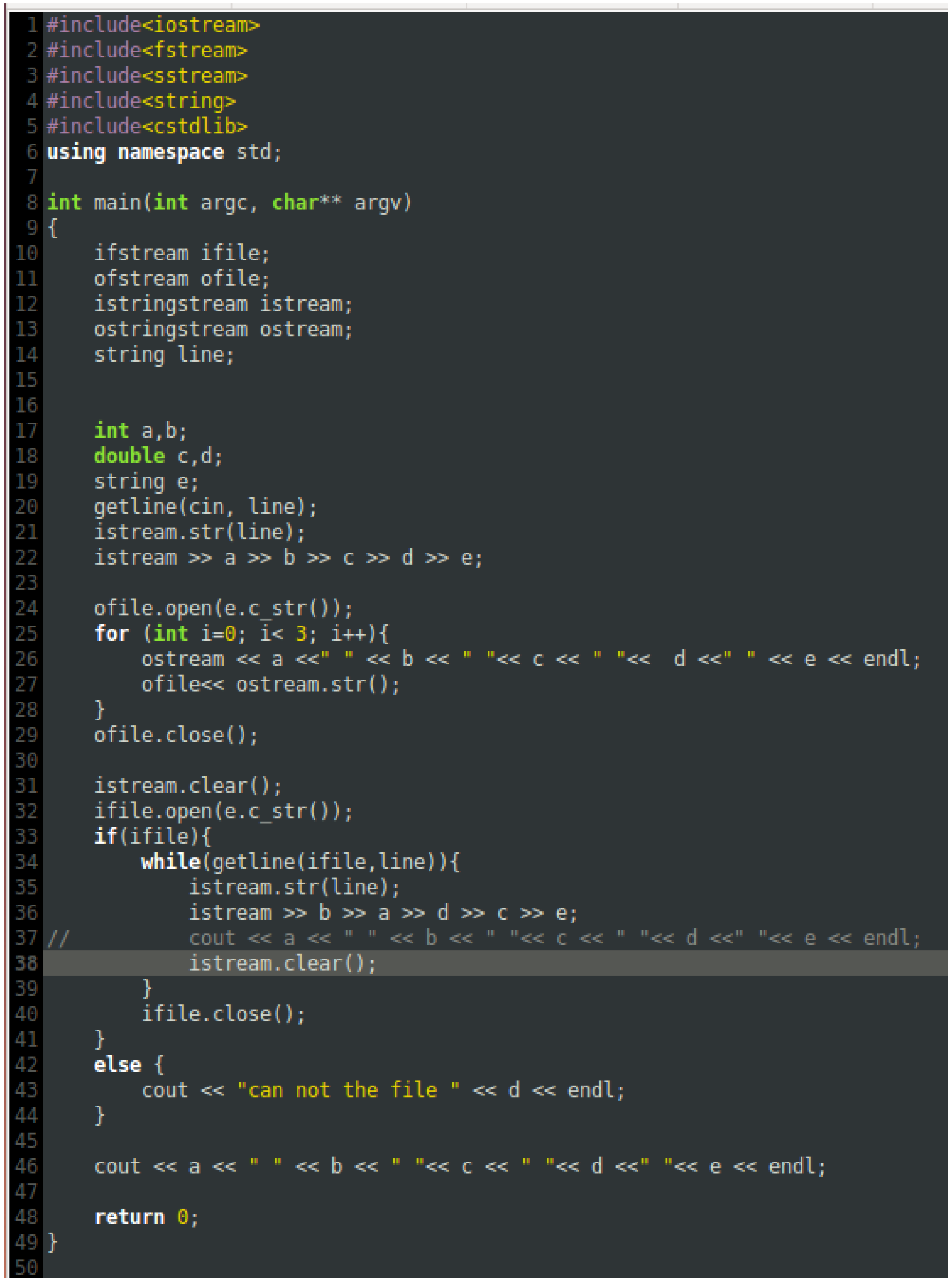
- C++ defines string stream in the sstream library

```
    #include <sstream>
    istringstream iStream;
    ostringstream oStream;
```

```
iStream.str(a_string);
String result = oStream.str();
```

String Stream

Example



```cpp
#include<iostream>
#include<fstream>
#include<sstream>
#include<string>
#include<cstdlib>
using namespace std;

int main(int argc, char** argv)
{
    ifstream ifile;
    ofstream ofile;
    istringstream istream;
    ostringstream ostream;
    string line;


    int a,b;
    double c,d;
    string e;
    getline(cin, line);
    istream.str(line);
    istream >> a >> b >> c >> d >> e;

    ofile.open(e.c_str());
    for (int i=0; i< 3; i++){
        ostream << a <<" " << b << " "<< c << " "<<  d <<" " << e << endl;
        ofile<< ostream.str();
    }
    ofile.close();

    istream.clear();
    ifile.open(e.c_str());
    if(ifile){
        while(getline(ifile,line)){
            istream.str(line);
            istream >> b >> a >> d >> c >> e;
//          cout << a << " " << b << " "<< c << " "<< d <<" "<< e << endl;
            istream.clear();
        }
        ifile.close();
    }
    else {
        cout << "can not the file " << d << endl;
    }

    cout << a << " " << b << " "<< c << " "<< d <<" "<< e << endl;

    return 0;
}

```

Outline

- Function Call Mechanism
- Categorizing Data: enum
- Passing Arguments to Program
- I/O Streams
  - File Stream
  - String Stream
- Testing
  - Specific test cases
  - Incremental testing
- Exception

Write Specific Tests

- What are examples of these cases for testing the power number in project 1?
  - A positive integer is called a power number if it equals m^n, where m and n are both integers and $n \geq 2$.

- Simple inputs: 125
- Boundary conditions: 1 20,000,000 (INT_MAX if not specified)
- Nonsense: -1 20,000,001 non-integer values

esting

What I recommend

rd each pa de(funct n) as unit

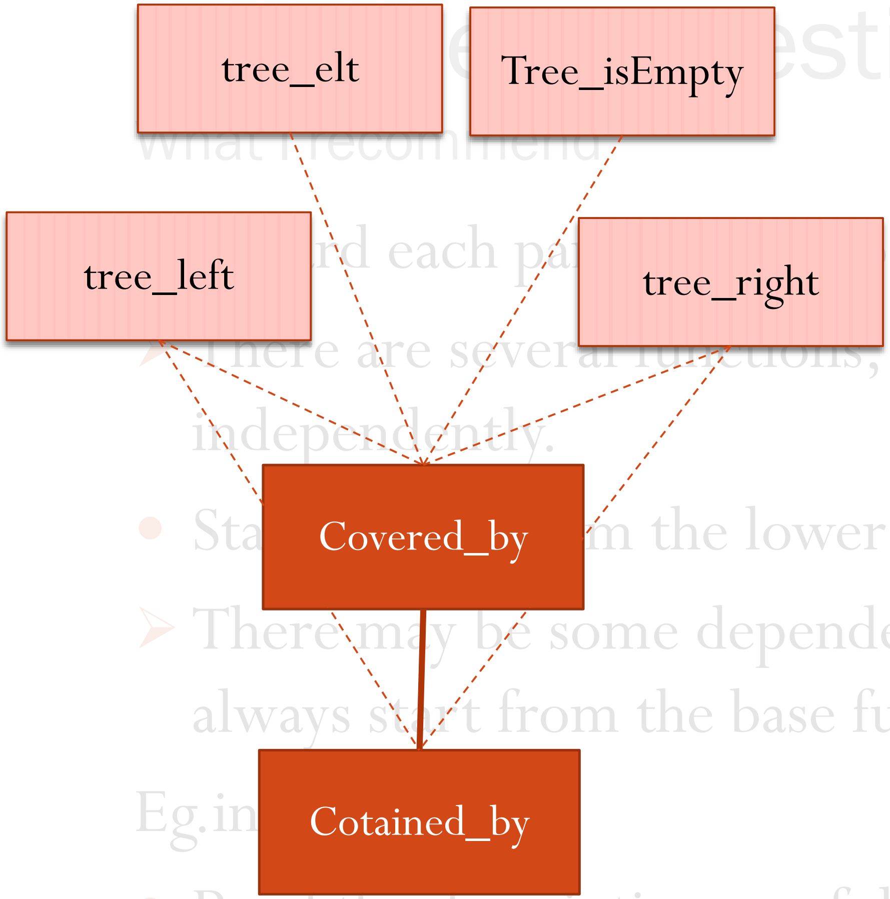
➢ There are several functions, first test each function independently.

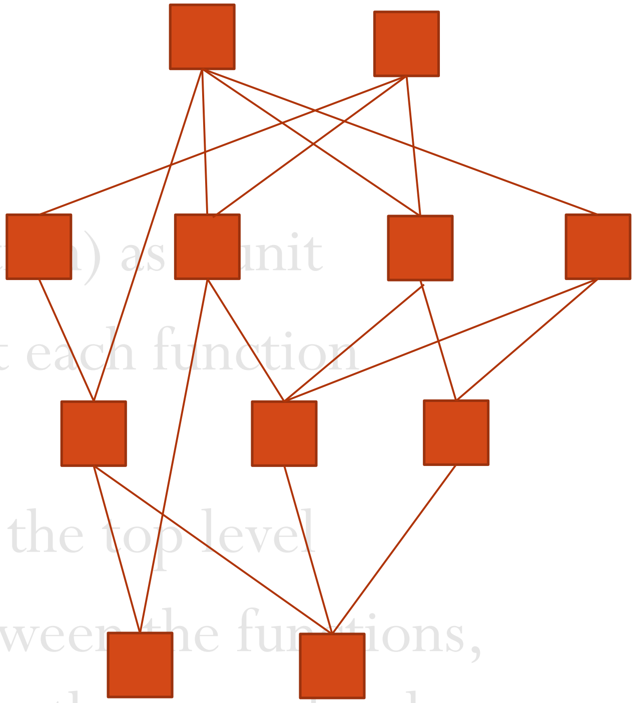
- Sta m the lower level to the top level

➢ There may be some dependency between the functions, always start from the base function to the upper level

Eg.in

- Read the description carefully

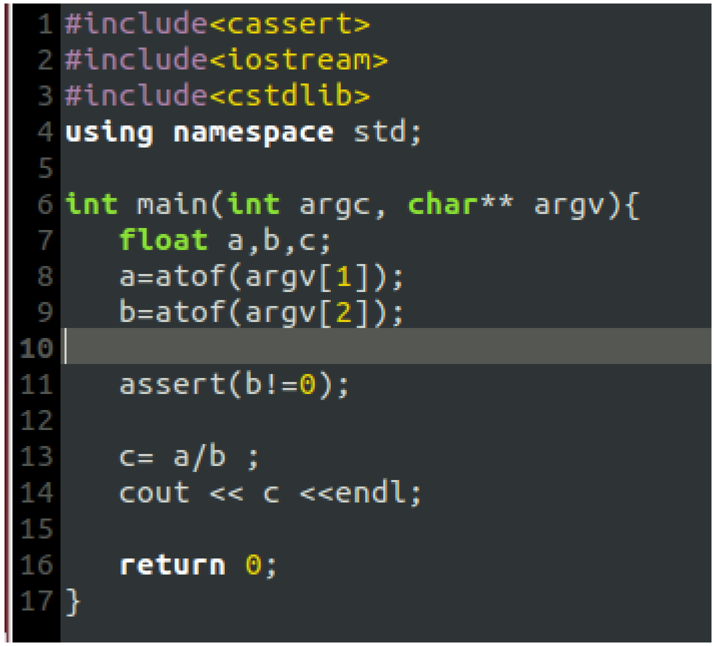




Using Assert Function

- `#include <cassert>`
- `assert` for the condition that should hold.

```
int smaller = min(a, b);
assert ((smaller == a && smaller <= b) || (smaller == b && smaller <= a)
);
```



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

int main(int argc, char** argv){
    float a,b,c;
    a=atof(argv[1]);
    b=atof(argv[2]);

    assert(b!=0);

    c= a/b ;
    cout << c <<endl;

    return 0;
}
```

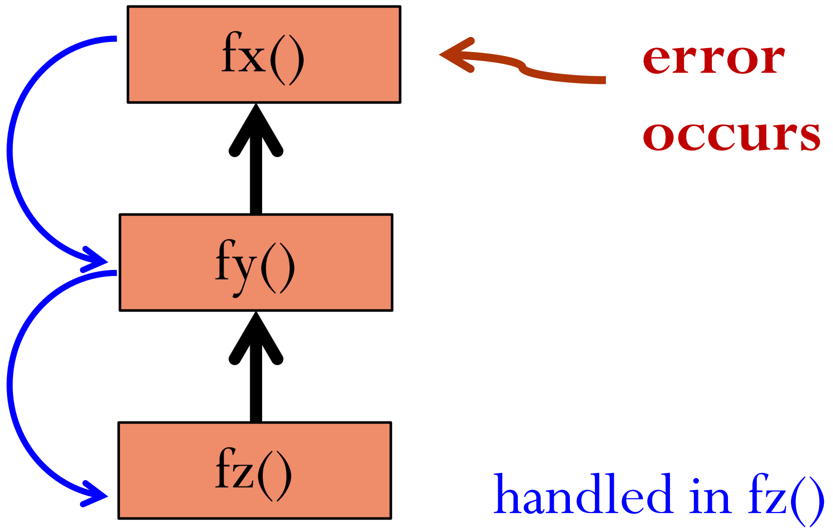
Outline

- Function Call Mechanism
- Categorizing Data: enum
- Passing Arguments to Program
- I/O Streams
  - File Stream
  - String Stream
- Testing
  - Specific test cases
  - Incremental testing
- Exception

Exceptions

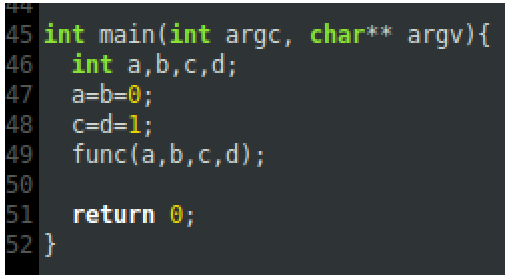
Dealing with runtime errors

- **Exception**: something bad that happens in a block of code, such as a bad parameter that prevents the block from continuing to execute.

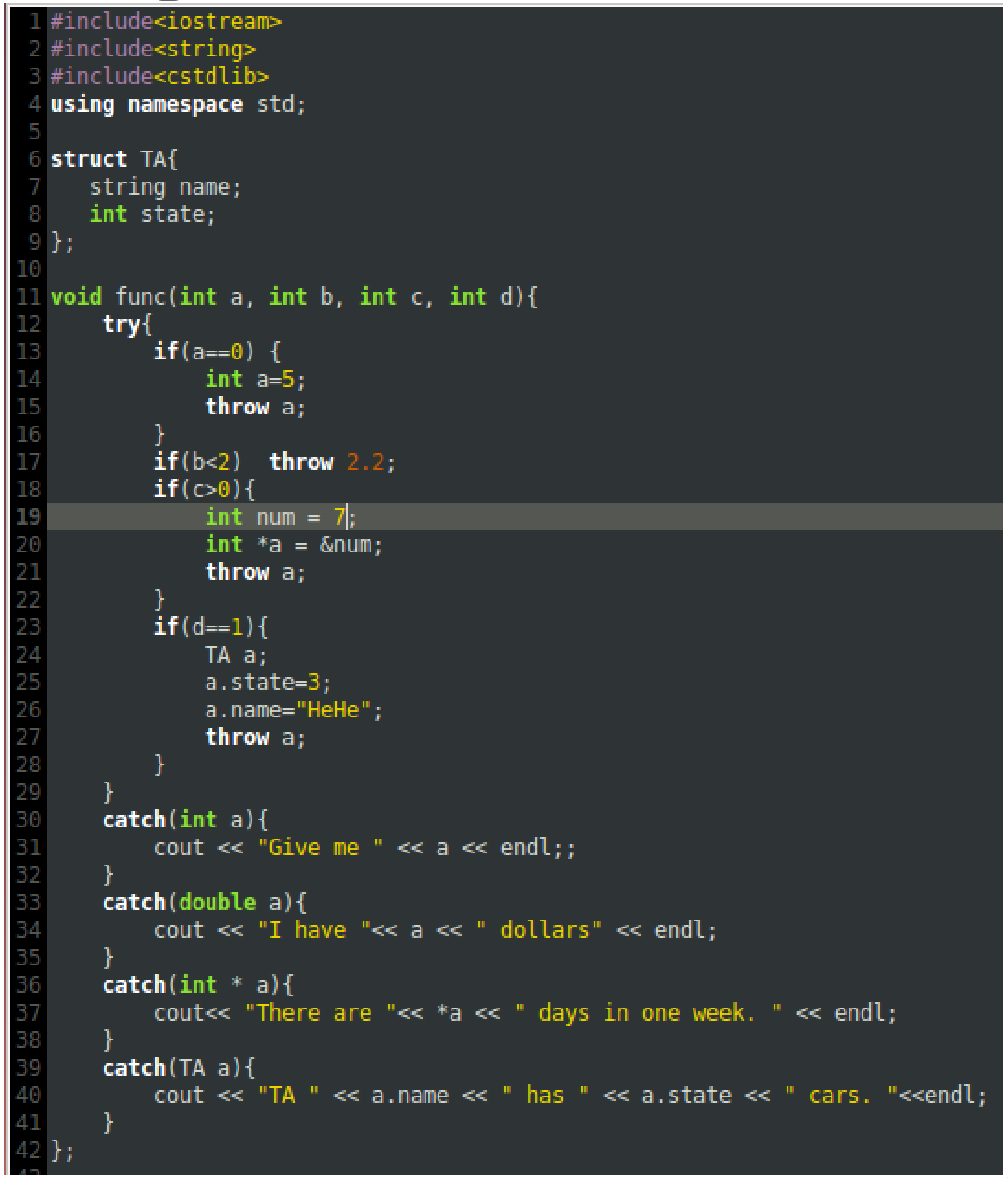


Exception Handling

Usage in C++



```cpp
int main(int argc, char** argv){
  int a,b,c,d;
  a=b=0;
  c=d=1;
  func(a,b,c,d);

  return 0;
}
```



```cpp
#include<iostream>
#include<string>
#include<cstdlib>
using namespace std;

struct TA{
   string name;
   int state;
};

void func(int a, int b, int c, int d){
    try{
        if(a==0) {
            int a=5;
            throw a;
        }
        if(b<2)  throw 2.2;
        if(c>0){
            int num = 7;
            int *a = &num;
            throw a;
        }
        if(d==1){
            TA a;
            a.state=3;
            a.name="HeHe";
            throw a;
        }
    }
    catch(int a){
        cout << "Give me " << a << endl;;
    }
    catch(double a){
        cout << "I have "<< a << " dollars" << endl;
    }
    catch(int * a){
        cout<< "There are "<< *a << " days in one week. " << endl;
    }
    catch(TA a){
        cout << "TA " << a.name << " has " << a.state << " cars. "<<endl;
    }
};
```

If you don’t know something about C++, first try it by your hand!

——Zhou Hongkuan