A Course on C++

Type Casting Mechanism

Type Casting Mechanism

- Casting – forced type conversion
- C style typecasting
  - a = (type) b;

```
int i = 65;
float a;
a = (float) i * 5;
```

- Draw back of C Style
  - Tracing is difficult
  - All type cast looks different

Casting in C++

- static_cast
- const_cast
- reinterpret_cast
- dynamic_cast

static_cast

- int i = 50;
  long l;
  float f;
  l = i;
  f = i;
- Better
  l= static_cast <long>(i);
  f = static_cast <float> (i);

const_cast

- ```
  const int a = 100;
  int * ptr  = (int *)  &a; // one way

  ptr = const_cast <int *> (&a);
  //better way
  ```
  ```

reinterpret_cast

- int i = 800;
  int * p = reinterpret_cast <int *> (i)

  i = reinterpret_cast <int > (p);

Run Time Type Information

```
base * p1, *p2;
derv1  d1, *pd1;
derv2  d2, *pd2;

pd1 = &d1;
pd2 = &d2;
p1 = &d1;
p2 = &d2;

cout << typeid(p1).name()
// gives the name of the class of
which p1 is pointing in this example it
```

- if(typeid(*p1) == typeid (*pd1))
  cout << “of same type” << endl;
  else
  cout << “not of same type”;
- Assignment will happen only if casting is successful
  if (pd1 = dynamic_cast < derv1 *> (p1))
  cout << “of type derv1”;
  else
  cout << “ not derv1 type”;

const

- Prefer consts over #defines
- char str[] = “hello”;
- char * p = str
  - Non-const pointer, non-const data
- const char *p = str;
  - Non const pointer, const data
- char * const p = str;
  - Const pointer , non const data
- const char * const p = str;
  - Const pointer, const data

const functions

- class ex
  {
    private: int data;
    public:
      sample(){ data = 0}
      void const_fun() const
      {
        data = 100;
      }
  };

Error

Inline Functions

- If a function is declared as inline, at the time of compilation, the body of the function is expanded at the point at which it is invoked
- For small functions it removes all the overheads of function calls
- To make a function inline , the key word inline is prefixed for a function
- Inline is a request for a compiler
- Function containing a loop may not be expanded

Inline vs. Macros

- In both the cases during the compilation time, the body of the macro or the function is expanded
- Inlines are preferred over the macros two main reasons

```
inline int square(int x)
{
        return (x*x);
}
```

Advantages of inline over macros

- The types of the arguments are checked against the parameter list in the declaration for the function
  - Hence any mismatch in the parameters can be detected at the time of compilation
  - This allows inline functions to be overloaded, which is not possible in the case of macros
- In certain situation macros does not behave in the same manner as a function call, which may lead to unpredictable results

Error Handling

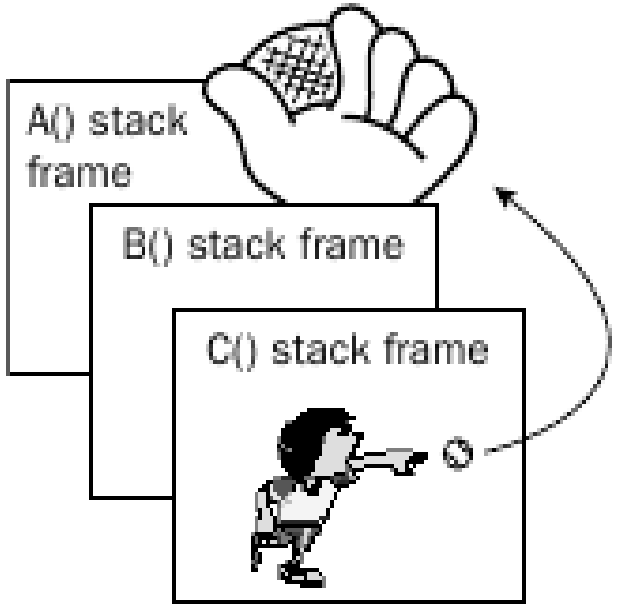
Types of Error

- Syntax
- Linker
- Run Time

Handling Runtime Errors

- Error code
  - Non Object Oriented
- Exceptions
  - Object Oriented

What are Exceptions



What are Exceptions?

- A mechanism for a piece of code to notify another piece of code of an error condition with out processing through the normal code paths
- The code that encounters the error throws the exception
- The code that handles the exception catches it

What are Exceptions? contd..

- When a piece of code throws an exception
- The program control immediately stops executing code step by step and transitions to the exception handler
- The exception handler can be any where in the program

Why Exceptions In C++ Are a Good Thing

- Run time errors in c++ programs are inevitable
- Error handling in most c and c++ programs is messy and ad hoc
- C style of error handling is not suitable for c++ programs
- The integer return codes and errno are used inconsistently
  - Some time -1 for failure and some times non zero

Why Exceptions In C++ Are a Good Thing

- There are several advantages of exceptions over ad hoc approaches in c
  - Return codes from functions can be ignored. Exceptions cannot be ignored
  - Integer return codes do not contain any semantic information
  - Different numbers can mean different things to different programmers
  - Exceptions can contain semantic information in both their type names and, if they are objects

Advntages contd...

- You can use exceptions to pass as much information as you want from the code that finds the error to the code that handles it
- Exceptions can also be used to communicate information other than errors
- Exceptions handling can skip levels of the call stack