RTTI and Advanced Type Cast

Run-Time Type Identification(RTTI)

- RTTI is used to dynamically determining an object's type
- Two RTTI operations:
  - typeid( ) function *(identification of types)*
  - dynamic_cast< > *(down casting)*
- Useful for type specialization in code
- Very useful for templates

typeid() function

- Returns a object of type_info describing that type
- typeid() can be used on any variable or type
- typeid().name() returns the type name as a string
- type_info‘s can be compared using the == and != operators
- It is “polymorphic-friendly”
- Must include the following header:

```
#include <typeinfo>
```

typeid() Example


```
#include<typeinfo>
#include<string>
#include<iostream>
using namespce std;
int main()
{ cout<<"\n Type="<<typeid(int).name();
cout<<"\n Type="<<typeid(string).name();
return 0;
}  // typeid() operator returns constant ref of type_info which has information about type.
```

typeid() Example


```
// Test if a given object is a basic numeric type
 template <class T>
 bool IsNumericType(T x)
 {
     if (typeid(x) == typeid(short)) return true;
     if (typeid(x) == typeid(long)) return true;
     if (typeid(x) == typeid(int)) return true;
     if (typeid(x) == typeid(double)) return true;
     return false;
 } // This function check type of variable
```

Advanced Type Casting

- C++ supports two types of type casting
  - Implicit Casting/Function call casting
  - Explicit Casting
- Implicit casting example:

```
int main()
{ float f =10;
  int i = (int) f;
  float f1 = (float) i;
  float f2 = float(200); // Function call cast
   return 0;
}
```

Advanced Type Casting

- Explicit Casting
  - There are 4 types of casting
    1. const_cast (for const to non-const and vice versa)
    2. reinterterpret_cast (for pointer conversion)
    3. static_cast (data type conversions)
    4. dynamic_cast (for down casting of polymorphic types)

const_cast example


```
#include<iostream>
using namespace std;
int main()
{ const int a=10;
  volatile int v = 20;
  int *sp = const_cast<int*>(&a);
  int *vp = const_cast<int*>(&v);
  cout<<"\n"<<*sp;
  cout<<"\n"<<ci;
  return 0;
}
```

reinterpret_cast example


```
#include<iostream>
using namespace std;
int main()
{ int  *ip  = NULL;
  int i = 650000;
  ip = &i;
  char *pc = reinterpret_cast<char*>(ip);
  cout<<"\n"<<*pc;
  return 0;
} //This cast works at bit level
```

static_cast example

- **It is used for promotion and Truncation**
- Conversion of void*

```
#include<iostream>
using namespace std;
int main()
{ long l = 100000;
  float f = 10.20F;
  int i = 10;
  l = static_cast<long>(i);
  f = static_cast<float>(i);
  i = static_cast<long>(l);
  cout<<"\n"<<i;
  cout<<"\n"<<f;
  return 0;
}
```

dynamic_cast example

- dynamic_cast< > is used to cast one polymorphic type to another type within its inheritance chain.
- dynamic_cast< > performs a safe “downcast”.
- dynamic_cast< > operations must be used

on **polymorphic pointers or references only**

Example: Consider a inheritance example where CricketPlayer, FootBallPlayer classes inherits Player class. Player class has virtual functions hence it becomes polymorphic type.

dynamic_cast example


```
#include<iostream>
#include<typeinfo>
using namespace std;
int main()
{ Player *pp= NULL;
 CricketPlayer c;
CricketPlayer *cp = NULL;
 FootBallPlayer f;
FootBallPlayer *fp = NULL;
 p = &c;
```

dynamic_cast example (cont ..)


```
    if(typeid(*pp) == typeid(CricketPlayer )){
     cp = dynamic_cast<CricketPlayer*>(pp);
    }
if( FootBallPlayer *ptr =  dynamic_cast<FootBallPlayer*>(pp)){
{ cout<<"\n Points to FootBallPlayer";
}
return 0;
} // Above example shows down casting
```

Thank You

Cast me carefully !!!!

……. Data Types