C++ proposal
version 2

New keyword for C++: dynamictype

Jerry Moravec

Unemployed, almost-homeless
Czech Republic
E-mail: j.moravec.email@seznam.cz

Abstract

The C++ language is here for more that 36 years and is still under development. Sometimes, C++ borrows some ideas from other programming languages, sometimes works on its own way. Presented proposal try to introduce a new type of basic variable which would be capable to hold a type of a variable or a type of a class or a struct. This variable should be capable to define target new type in type-cast operations such us: static_cast or dynamic_cast etc. Both at running time and at compilation time. Type of a variable or a class/struct is usually given by its unique name. Thus, the new basic variable should contain a string – the name of basic datatype or a type of user defined datatype. This name is then used in the type-cast operation at running time. The target type is, of course, well known at compilation time.

1. Description

Associated operator with type-cast operations in C++ is usually given inside the angle brackets "<target_type>" or in the round brackets "(target_type)". The type-cast operation which uses the angle brackets is connected with several keywords static_cast, dynamic_cast and reinterpret_cast. Proposed new key-word should be capable to work with both cases of type-case operations.

I would like to propose a new key-word of the C++ language: "dynamictype". It is useful in type-cast operation(s) such us:

```
int myvariable = 0;
dynamictype mytype;
mytype = typeof(float);
std::cout << dynamic_cast< mytype >(myvariable);
```

The "mytype" is not a standard variable, nor pointer. This variable holds "a type definition", the float type in this case. It is the type-casting extension. E.g. "dynamic_cast" has, indeed, in the angle brackets "a variable" which is "a type" not a standard variable such us int or float or pointer. The "`mytype`" is a variable, but the type stored into it is well known at compilation time.

2. Motivation

Main motivation is a code simplification. The best is a real example, so complex example is listed below. The main important part is on the lines 20 and 50-62.

```
1.  #include <iostream>
2.  #include <string>
```

```
3.   #include <iterator>
4.   #include <algorithm>
5.   #include <vector>
6.
7.   struct Example_1 {
8.
9.       int item1;
10.      float item2;
11.  };
12.
13.  struct Example_2 {
14.      unsigned long item1;
15.      int item2;
16.  };
17.
18.  struct arrItem {
19.      int key;
20.      dynamictype Decl;
21.      void* link;
22.  };
23.
24.
25.  int main()
26.  {
27.
28.      arrItem *item = nullptr;
29.
30.      Example_1 *e1 = nullptr;
31.      Example_2 *e2 = nullptr;
32.
33.      std::vector<arrItem> DataList;
34.
35.      // data type 0 is Example_1
36.      e1 = new Example_1();
37.      e1->item1 = 101;
38.      e1->item2 = (float)102.10;
39.
40.      DataList.push_back(arrItem());
41.      DataList[0].key = 0;
42.      DataList[0].typeDecl = typeof( Example_1 );
43.      DataList[0].link = e1;
44.
45.      // data type 1 is Example_2
46.      e2 = new Example_2();
47.      e2->item1 = 201;
48.      e2->item2 = (int)202;
49.
50.      DataList.push_back(arrItem());
51.      DataList[1].key = 1;
52.      DataList[1].typeDecl = typeof( Example_2 );
53.      DataList[0].link = e2;
54.
55.      for (int i = 0; i < (int)DataList.size(); i++) {
56.
57.          std::cout << "Example_1: key: "
58.                    << DataList[i].key
59.                    << "; item1: "
60.                    << dynamic_cast< DataList[i].typeDecl >( DataList[i].link )->item1;
61.                    << "; item2: "
62.                    << dynamic_cast< DataList[i].typeDecl >( DataList[i].link )->item2;
63.          std::cout << "\n";
64.
65.      }// for
66.
67.
68.      delete(DataList[0].link);
69.      DataList[0].link = nullptr;
70.      delete(DataList[1].link);
71.      DataList[1].link = nullptr;
72.
73.      DataList.clear();
74.
75.
76.  }// int main()
```

If I do not use the expression:

“`dynamic_cast< DataList[i].typeDecl >( DataList[i].link )->item1;`”

I have to use a switch-case key-word. Identical example listed above needs to be written is this standard way utilizing switch-case in this way:

```
#include <iostream>
#include <string>
#include <iterator>
#include <algorithm>
#include <vector>

struct Example_1 {

    int item1;
    float item2;
};

struct Example_2 {
    unsigned long item1;
    int item2;
};


struct arrItem {
    int key;
    void *link;
};


int main()
{

    arrItem *item = nullptr;
    Example_1 *e1 = nullptr;
    Example_2 *e2 = nullptr;

    std::vector<arrItem> DataList;

    // data type 0 is Example_1
    e1 = new Example_1();
    e1->item1 = 101;
    e1->item2 = (float)102.10;

    DataList.push_back(arrItem());
    DataList[0].key = 0;
    DataList[0].link = e1;

    // data type 1 is Example_2
    e2 = new Example_2();
    e2->item1 = 201;
    e2->item2 = (int)202;

    DataList.push_back(arrItem());
    DataList[1].key = 1;
    DataList[1].link = e2;


    for (int i = 0; i < (int)DataList.size(); i++) {

        switch ((int)DataList[i].key)
        {
            case 0:
            {
                Example_1 *ex1 = (Example_1*)DataList[i].link;
                std::cout << "Example_1: key: "
                          << DataList[i].key
                          << "; item1: "
                          << ex1->item1
                          << "; item2: "
                          << ex1->item2;
                std::cout << "\n";

                break;

            }// case 0:
            case 1:
            {
                Example_2* ex2 = (Example_2*)DataList[i].link;
                std::cout << "Example_2: key: "
                          << DataList[i].key
```

```
                    << "; item1: "
                    << ex2->item1
                    << "; item2: "
                    << ex2->item2; // e.g. safe_cast<string>(ex2->item2);
            std::cout << "\n";

            break;
        }// case 1:

    }// switch

}// for
```

3. Consequences

The consequences for compiler-makers are, unfortunately, really wide, but the illustrative simplified code is above. The compiler will be more complicated 😊. However, the code will be more simplified.

4. Experience

The practical example is described in the section 2. of this proposal, but such improvement would be great "gadget" not only for C++ but for many others programming languages such us C# or C++/CLI (ECMA 327) or C++/WinRT etc.

5. Summary

This paper proposed the extension of C++ with the keyword „dynamictype", the keyword enables to change a data-type using classic type-cast operation even inside angle brackets of standard type-case expressions such as dynamic_cast, reinterpret_cast. The main advantage of such new thing is a code simplification. The implementation cost is maybe not "small", but the consequences for existing code are minor or none.