ICE passing args and pointers to functions.cpp

```
1 //============================================================================
2 // Name        : ICE.cpp
3 // Author      : Paul Hrycewicz
4 // Version     :
5 // Copyright   : Your copyright notice
6 // Description : Hello World in C++, Ansi-style
7 //============================================================================
8
9 #include <iostream>
10 using namespace std;
11 // func1 - passing an int by value
12 // func2 - passing an int by reference
13 // func3 - passing an int array
14 // func4 - trying to pass an array by reference - can't
15 // func5 - passing a pointer to an int, and changing what it points to
16 // func6 - passing a pointer to an int, allocating a new int
17 // func7 - same as func 6 but passing the pointer by reference
18 // func8 - trying to treat a passed pointer as an array
19 // func9 - treating a passed pointer as a pointer to an array, and
20 //          manipulating the address in the pointer
21 // func10 - coding as if an array of pointers was passed
22 // func11 - passing a pointer to an array, and accessing the array
23 //          "normally"
24
25 void func1(int a)
26 {
27     a++;            // this changes only the local copy
28     return;
29 }
30 void func2(int &a)
31 {
32     a++;
33     return;         // this changes the "real" copy
34 }
35 void func3(int ar[])
36 {
37     ar[0]++;        // this changes the "real" copy since an array is
38                     // automatically passed by reference
39     return;
40 }
41 /*  can't do this - makes no sense to pick up &ar
42 void func4(int &ar[])
43 {
44     ar[0]++;
45     return;
46 }
47 */
48 void func5(int * p)
49 {
50 //  *p++;           // be careful = this increments the pointer p,
51                     // but it's just fine and dandy as far as the compiler
52                     // is concerned
53     *p=*p+1;        // this increments what p points at
54     return;
55 }
56 void func6 (int * p)
57 {
```

```
58      p = new int;     // replace whatever p pointed at with a new one
59      *p = 10;         // and this sets the new int to 10
60      return;
61  }
62  void func7 (int * & p)  // passing an int pointer by reference
63                          // if we pass the pointer itself by reference,
64                          // we can change the "real" copy
65                          // of the pointer itself
66  {
67      p = new int;        // point p at a brand new int
68                          // this change will be seen by the caller
69      *p = 11;            // and this is a standard dereference
70      return;
71  }
72  /* can't do this, p is not an array
73  void func8 (int * p)
74  {
75      *p[0]=*p[0]+1;      // the compiler won't let us get away with this
76      return;
77  }
78  */
79  // we can do this, adjusting the pointer to get to the
80  // array element we want
81  void func9 (int * p)
82  {
83      cout << "in func9 0th element of array is " << *p << endl;
84      p++;
85      cout << "in func9 1st element of array is " << *p << endl;
86                  // we never reset p after incrementing it,
87                  // but that's OK, it's passed by value so
88                  // the caller won't see that change we made
89      return;
90  }
91  // we can do the following, passing an array of pointers
92  // but be careful, this is different than an array of ints
93  void func10 (int * p[])
94  {
95      *p[4] = *p[4] + 1;  // adds 1 to the int pointed to by
96                          // the 5th element of the pointer array
97      p[5]=p[5]+1;        // this increments (by the length of an int)
98                          // the sixth element of the array of pointers
99      return;
100 }
101 void func11 (int * ar)
102 {
103     ar[0]++;
104     return;
105 }
106
107
108 int main() {
109     int a = 1;
110     int ar[10];
111     ar [0] = 2;
112
113     func1(a);
114
```

```
115     func2(a);
116
117     func3(ar);
118
119     int * p;
120     p = new int;
121     *p = 5;
122     func5(p);
123
124     func6(p);
125
126     func7(p);
127
128     p = new int[10];     // an array of ints
129     func9(p);
130
131     int * pp[10];        // an array of pointers
132                          // 10 int pointers pointing somewhere...to
133                          // 10 different ints
134     func10(pp);
135
136     func11(p);           // p is a pointer to an array of ints
137
138     return 0;
139 }
140
```