深入理解成员指针

成员指针

- 实例成员指针(数据成员和函数成员)
- 静态成员指针(数据成员和函数成员)
- 深入理解实例成员指针

1. 实例成员指针 (1)

(1) 定义实例成员指针

实例数据成员指针：

变量类型 类名::*变量名；

实例函数成员指针：

返回类型 (类名::*变量名)(参数原型)；

- 实例数据成员指针的值是数据成员的偏移值；
- 实例函数成员指针实际上是函数的物理地址。

1. 实例成员指针 (2)


```
struct  A {
      int  j, i;
      int  f(int x) { return i + x; }
      int  A::*u;
      int  (A::*pf)(int);
};

int  main( ) {
     A  a;
     a.u = &A::i;                    //a.u = 4
     a.pf = &A::f;                   //a.pf = A::f( )的物理地址
     int  A::*p = &A::i;             //p = 4
     int  (A::*ff)(int) = &A::f;     //ff = A::f( )的物理地址
};
```

1. 实例成员指针 (3)

(2) 使用实例成员指针

必须结合对象(引用)使用 .*、或结合对象指针使用 ->*，来访问对象中的成员：

- 对象.*数据成员指针
- 对象指针->*数据成员指针
- (对象.*函数成员指针)(参数)
- (对象指针->*函数成员指针)(参数)

1. 实例成员指针 (4)


```
struct  A {
    int  j, i;
    int  f(int x) { return i; }
    int  A::*u;
    int  (A::*pf)(int);
} a;
```

```
int  main( ) {
    A *p = &a;
    a.u = &A::i;      //a.u = 4
    a.pf = &A::f;   //a.pf=A:f()的物理地址
    int  A::*u = &A::i;  //u = 4
    int  (A::*ff)(int) = &A::f;  //ff=物理地址
    int  i = a.*u;
    i = p->*u;
    i = a.*a.u;
    i = p->*a.u;
    int  x = (a.*ff)(1);
    x = (p->*ff)(2);
    x = (a.*a.pf)(3);
    x = (p->*a.pf)(4);
};
```

1. 实例成员指针 (5)

(3) 实例成员指针说明

- 实例数据成员指针是数据成员相对于对象首地址的偏移。
- 实例函数成员指针是函数成员的物理地址。
- 实例成员指针：不能移动；
  不能参与运算；
  不能强制类型转换。

1. 实例成员指针 (6)


```
struct  A {
    int  j, i;
    int  f(int x) { return i; }
    int  A::*u;
    int  (A::*pf)(int);
} a;
```

```
int  main( ) {
    int A::*pi = &A::i;
    int (A::*pf)(int) = &A::f;
    pi++;  //错, 不能移动
    pf++;  //错, 不能移动
    pi = (int A::*)i;    //错, 不能强制转换
    int k1 = pi + 1;     //错, 不能参与运算
    int k2 = (int)pi;    //错, 不能强制转换
    int k3 = (int)pf;    //错, 不能强制转换
    int (*f)( ) = &main;
    k3 = (int)f;  //对, 普通指针可以强制转换
    float i = (float)(a.*pi);       //对
    float j = (float)(a.*pf)(1);  //对
};
```

2. 静态成员指针 (1)

- 静态成员指针本质上是普通指针，需要以普通指针的形式去定义和访问，不能使用“类名::*变量名”实例成员的方式去定义和访问。
- 静态成员指针的值是静态成员的物理地址。

2. 静态成员指针 (2)


```
struct  A {
    int i, j;
    static int k;
    int f(int x) { return 0; }
    static int g(int x) { return x; }
} a;
int A::k = 1;
```

```
int  main( ) {
    int A::*pi = &A::k;            //错
    int (A::*pf)(int) = &A::g;  //错
    int *p = &A::k;
    int (*f)(int) = A::g;
    float *q = (float *)f;    //对
    int k1 = (*f)(100);
    p++;                                 //对
    q++；q + 1;                      //对
    f++;                                 //错
    int k2 = f + 1;                  //错
    int k3 = (int)f + p[2];  //对
};
```

3. 深入理解函数成员指针 (1)

- 类的函数成员的代码是不依赖于任何对象而独立存在的，实例函数成员指针实际上是成员函数的物理地址。
- 能否不通过对象而直接调用实例函数成员呢？
  (不通过“(a.*f)(实参)”的形式去调用实例成员函数)

3. 深入理解函数成员指针 (2)


```
struct  A {
   int  a = 1;
   int  f( ) { return 1; }
   int  g(int x) { return a + x; }
} a;


//不通过对象去调实例函数成员
int main( )
{   int  (A::*h)( ) = &A::f;
    int  k = (a.*h)( );
    int  (*p)(void *);
    p = (int (*)(void *))h;  //error
    k = (*p)((void *)0);  //collapse
}
```

实例函数成员指针实际上是物理地址，但不能进行强制类型转换和指针运算！

```
//不通过对象去调实例函数成员
int main( )
{   int  (A::*h)( ) = &A::f;
    int  k = (a.*h)( );
    int  (*p)(void *);
    //p = (int (*)(void *))h; //error
    _asm mov eax, h
    _asm mov p, eax
    k = (*p)((void *)0);   //k=1
}
```

3. 深入理解函数成员指针 (3)


```
struct  A {
   int  a = 1;
   int  f(int x)
        { return 2 * x; }
   int  g(int x)
        { return a + x; }
} a;
```

不通过对象去调
实例函数成员

```
int main() {
    int (A::*pf)(int) = &A::f;
    int (A::*pg)(int) = &A::g;
    int (*f)(void *, int);
    int (*g)(void *, int);
    //f = (int (*)(void *, int))pf;   //error
    //g = (int (*)(void *, int))pg;  //error
    _asm mov eax, pf
    _asm mov f, eax
    _asm mov eax, pg
    _asm mov g, eax
    int k1 = (*f)((void *)0, 10);   //k1 = 20
    int k2 = (*g)((void *)0, 10);   //collapse
    int k3 = (*g)(&a, 10);  //k3 = 11
}
```

The end.