Table of Contents

فانکشن‌های C++ چه چیزی را مینویسند و فراخوانی می‌کنند.

چه موقع یک کلاس خالی یک کلاس خالی نیست؟ وقتی که C++ وارد میدان می‌شود. وقتی شما چیزی رو تعریف نکنید، خود کامپایلر ورژن خودش رو از کپی سازنده، اپراتور انتساب، و destructor رو اعلان می‌کنه، اگر شما کلا هیچ constructorای رو اعلان نکرده باشید در این صورت کامپایلر یک سازنده پیش‌فرض برای شما ایجاد می‌کنه. همه‌ی این توابع به صورت public و inline خواهند بود. در نتیجه، اگر شما یک همچین چیزی رو بنویسید:

```
class empty
{

};
```

این در واقع مشابه این هست که شما کد رو به صورت زیر می‌نوشتید:

```
class Empty
{
public:
   Empty(){}                 //default constructor
   Empty(const Empty& rhs){}    //copy constructor
   ~Empty(){}                 //destructor
   Empty& operator =(const Empty& rhs){}  //copy assignment operator
};
```

این توابع در صورت نیاز بهشون تولید میشند، اما این که بهشون نیاز داشته باشیم خیلی ساده‌ست، کدهای زیر باعث تولید هر کدام از توابع بالا میشند:

```
   Empty e1;  //default constructor
   Empty e2(e1);  // copy constructor
   e2=e1;  //copy assignment
```

این که اجازه بدهیم که کامپایلر توابع را برای ما بنویسد، در این صورت توابع دقیقا چه کاری برای ما انجام خواهند داد؟ سازنده پیش فرض و یک نابودگر اولیه در واقع توسط کامپایلر جاگذاری می‌شود. توجه داشته باشید که نابودکننده به صورت non-virtual می‌باشد(آیتم ۷ رو ببینید)، مگر این که این کلاس از یک کلاس دیگر ارث بری کرده باشد که در اون کلاس یک destructor به صورت virtual تعریف شده باشد.

همچنین برای copy constructor و اپراتور copy assignment ، کامپایلر تمام داده‌های عضو non-static رو در سورس مربوط به شیء نهایی کپی می‌کنه. به طور مثال، یک template به نام NamedObject رو در نظر بگیرید که به شما اجازه میده که با اشیاء نوع T کار کنید:

```
template<typename T>
class NamedObject{
  public:
    NamedObject(const char* name,const T& value);
    NamedObject(const std::string& name,const T& value);
private:
    std::string nameValue;
    T objectValue;
};
```

چون یک سازنده در namedObject اعلان شده، کامپایلر constructor پیش‌فرض رو در این مورد تولید نمی‌کنه. این خیلی مهمه. این یعنی که اگر شما با دقت بیشتری یک کلاس رو طراحی کنید، که در اون سازنده‌ای وجود داشته باشه که آرگومان بگیره، در این صورت نیاز نیست نگران این باشیم که کامپایلر دوباره بخواد سازنده کلاس رو دوباره بسازه، که حتی هیچ آرگومانی هم نپذیره.

NamedObject نه کپی سازنده و نه اپراتور انتساب رو اعلان کرده، در این صورت خود کامپایلر دست به کار میشه و این توابع رو تولید می‌کنه(البته یادتون باشه که در صورتی که نیاز باشه این کار رو انجام میده). به کد کپی سازنده زیر نگاه کنید. نحوه‌ی تعریف سازنده کلاس در پیاده سازی به صورت زیر خواهد بود.

```
template<typename T>
NamedObject<T>::NamedObject(const char *name, const T &value)
{
    nameValue=name;
    objectValue=value;
}
```

```
    NamedObject<int> no1("Smallest Prime number",2);
    NamedObject<int> no2(no1);   //calls copy constructor
```

کپی سازنده‌ای که توسط کامپایلر تولید میشود، می‌بایست no2.nameValue و no2.ObjectValue را توسط no1.nameValue و no1.ObjectValue آغازدهی کند. نوع nameValue رشته خواهد بود، و نوع رشته استاندارد کپی سازنده دارد، در این صورت no2.nameValue با فراخوانی کپی سازنده string آغازدهی یا initialize می‌شود، دقت کنید آرگومان کپی سازنده string همان no1.nameValue خواهد

بود. از طرف دیگه، نوع NamedObject<int>::ObjectValue مشخصا int خواهد بـود، و int یـک نـوع built-in هست، در این صورت no2.ObjectValue با کپی کردن بیتی no1.ObjectValue شروع به کار و یا initialize خواهد شد.