بنام خدا

چگونه کدهای C++ خود را مستند و سازمان‌دهی کنیم ؟

معرفی

سبک برنامه‌نویسی در مورد این است که چگونه کدهای خود را مستند و سازماندهی کنید . سبک ورای زیبایی شناسیست . یک برنامه نوشته شده به سبک ثابت برای خواندن، درست کردن و حفظ کردم راحت‌تر است .
یک برنامه یک‌بار نوشته می‌شود ولی بارها خوانده می‌شود :

- درطول اشکال‌زدایی
- وقتی چیزی به برنامه اضافه می‌کنید
- وقتی کدی را بروز می‌کنید
- وقتی سعی می‌کنید کد را بفهمید

هر چیزی که باعث شود برنامه قابل خواندنتر و قابل فهم تر بشود زمان زیادی را صرفه جویی می‌کند حتی در یک اجرای کوتاه و یا وقتی برنامه کوچک است.
بیشتر برنامه‌نویسان معتقدند که کدنویسی استاندارد بسیار مهم است . مشکل اصلی این است که یک استاندارد واحد برای C++ وجود ندارد . به عنوان یک برنامه‌نویس حرفه‌ای شما باید برای تغییر استایل خود در جهت استاندارد کمپانی خود یا پروژه مورد نظر آماده باشید .
این دوره استانداردهای معمول صنعت برنامه‌نویسی را دنبال می‌کند . بعضی بخش‌های این دوره بر تخصیص‌های برنامه‌نویسی تأکید دارد و مرتبط با استایل کدنویسی شماست که در زیر آورده شده است .

نامگذاری

1. از اسامی معنی‌دار استفاده کنید .

نامی را انتخاب کنید که مشخص کننده هدف شما باشد . نام خوب کمک می‌کند بفهمید در حال حل چه مشکلی هستید .

2. نام متغیرها

دو روش معمول برای استفاده وجود دارد که ممکن است شما از آن‌ها استفاده کنید . به هر حال باید توجه داشته باشید که فقط و فقط از یک روش در یک برنامه استفاده کنید . بزرگان برنامه نویسی روش اول را ترجیه میدهد .

۱. استفاده از حرف کوچک برای شروع و استفاده از حروف بزرگ به عنوان جدا کننده (از آندرلاین(_) استفاده نکنید)

```
int myValue;
```

۲ . استفاده تمام از حروف کوچک و استفاده از آندرلاین (_) به عنوان جدا کننده

```
int my_value;
```

3. نام مقدارهای ثابت

برای مقدارهای ثابت باید از حروف بزرگ استفاده کنید و برای جدا کننده از آندرلاین (_) استفاده کنید

```
const int MY_CONST = 1;
```

4. نام توابع

دو روش معمول برای نامگذاری توابع وجود دارد . به هر حال شما باید دقت کنید که فقط و فقط از یک روش در یک برنامه استفاده کنید . استادان برنامه‌نویسی روش اول را ترجیه میدهند .

۱. روش اول استفاده از حرف کوچک برای شروع و حروف بزرگ به عنوان جدا کننده است

```
int myFunction();
```

۲. در این روش تماماً از حروف کوچک استفاده شده و از آندرلاین (_) برای جدا کننده استفاده می‌شود

```
int my_function();
```

5. نام کلاس‌ها

برای کلاس باید از حرف بزرگ برای شروع استفاده کرد و همینطور از حروف بزرگ به عنوان جدا کننده استفاده می‌شود .(از آندرلاین (_) استفاده نکنید)

```
class MyClassName
```

کامنت (Comments)

کامنت برای برنامه نویسانی است که کد را میخوانند . با نامگذاری خوب نیاز به کامنت گذاری به حداقل میرسد . تنها مورد استفاده از بلاک‌های کامنت اول هر فایل و قبل از تعریف هر تابع است .بلاک‌های کامنت در این بخش از دوره پوشش داده می‌شوند .

جای دیگری که از کامنت بجز بلاک‌های کامنت استفاده می‌شود ، وقتی کد شما غیر معمول و یا مبهم است وقتی چیزی مهم است و مبهم است ، کامنت میطلبد.

1. بلاک‌های کامنت (block comments)

Doxygen یک ابزار است که کامنت‌های داخل کد را مورد بررسی قرار میدهد و به صورت خودکار برای شما مجموعه‌ای از صفحات html ایجاد می‌کند . Ccdoc ابزار دیگری برای این کار است . صفحات تولید شده با این ابزارها کد شما را برای برنامه نویسان دیگر توضیح می‌کند برای دیدن مستندات تولید شده با این ابزار به Introduction مراجعه کنید .

مستندات نشأت گرفته از بلاک‌های کامنت است که شما مانند زیر تولید کرده‌اید :

- خط اول را مانند زیر دندانه‌دار کنید .
- کامنت را با نماد شروع کامنت شروع کنید. یک اسلش (/) و دو ستاره بلافاصله بعد از آن (**) مانند کد صفحه بعد

- هدف تابع یا کلاس را شرح دهید .
- یک خط خالی بین توضیحات و لیست برچسب‌ها بگذارید مانند کد زیر
- برای هر برچسب خط جدیدی ایجاد کنید
- خط پایانی با نماد انتها یعنی یک ستاره و یک اسلش به پایان میرسد دقت کنید که برای انتها فقط یک ستاره میگذاریم

```
/**
CS-11 Asn 2, checks.cpp
Purpose: Calculate the total of 6 checks

@author Ed Parrish
@version 1.1 01/03/17
*/
```

- برای اطلاعات بیشتر قسمت اطلاعات برچسب‌های سایت CcDoc را در Directives را ببینید .

2. بلاک کامنت فایل

هر فایل باید یک بلاک کامنت در بالای خود داشته باشد که شامل تعداد دوره ، تعداد انتساب‌ها ، نام فایل و هدف فایل باشد . معمولاً یک یا دو خط برای توضیح هدف کافیست . بعلاوه شما نیاز دارید برچسب author را بعلاوه اسم خودتا بنویسید و همینطور برچسب version که شامل ورژن و تاریخ تغییر در کد . برای مثال

```
/**
  CS-11 Asn 2
  checks.cpp
  Purpose: Calculates the total of 6 checks

  @author Ed Parrish
  @version 1.1 4/10/16
*/
```

برچسب‌های زیر باید همیشه استفاده شوند:

- @author
- @version

3. بلاک کامنت تابع

هر تابع قبل از تعریف یک بلاک کامنت نیاز دارد مانند زیر

```
/**
   Encodes a single digit of a POSTNET "A" bar code.

   @param digit the single digit to encode.
   @return a bar code of the digit using "|" as the long
   bar and "," as the half bar.
*/
string encode(int digit);
```

خط اول توضیحات و نحوه استفاده تابع است .

و مانند بالا باید حتماً از تگ های

- @param
- @return

استفاده شود .

4. یک مثال از مستند سازی برنامه

در زیر یک مثال ساده و کوچک از یک کد مستند شده می‌بینیم

```
/**
    CS-11 Asn 6
    sphere.cpp
    Purpose: Calculates the area of a circle and the volume
    of a sphere.
    @author Ed Parrish
    @version 1.0 3/17/04
*/
#include <iostream>
#include <cmath>

using namespace std;

const double PI = 3.14159;

/**
   Returns the area of a circle with the specified radius.
```

```
   @param radius The radius of the circle.
   @return The area of the circle.
*/
double area(double radius);

/**
   Returns the volume of a sphere with the specified radius.

   @param radius The radius of the circle.
   @return The volume of the sphere.
*/
double volume(double radius);

// Controls operation of the program.
int main(void) {
   double radius_of_both, area_of_circle, volume_of_sphere;

   cout << "Enter a radius to use for both a circle\n"
        << "and a sphere (in inches): ";
   cin >> radius_of_both;

   area_of_circle = area(radius_of_both);
   volume_of_sphere = volume(radius_of_both);

   cout << "Radius = " << radius_of_both << " inches\n"
        << "Area of circle = " << area_of_circle
        << " square inches\n"
        << "Volume of sphere = " << volume_of_sphere
        << " cubic inches\n";

   return 0;
}

// Returns the area of a circle with the specified radius.
double area(double radius) {
   return (PI * pow(radius, 2));
}

// Returns the volume of a sphere with the specified radius.
double volume(double radius) {
   return ((4.0 / 3.0) * PI * pow(radius, 3));
```

```
}
```

براکت‌ها

مبحث پرشور بسیاری از محافل برنامه نویسی نحوه چگونگی قرار دادن براکت است . هر دو روش در ادامه آمده قابل قبول است :

1. در همان خط کلمات کلیدی تعریف در انتهای خط

```
if (condition) {
    ...
}

while (condition) {
    ...
}
```

2. قرار دادن براکت شروع در زیر خط کلمات کلیدی مانند زیر

```
if (condition)
{
    ...
}

while (condition)
{
    ...
}
```

3. روش اول برای برنامه نویسان سنتی یونیکس و برنامه نویسان سی است بیشتر برنامه نویسان سی پلاس پلاس استایل استراستروپ را ترجیه میدهند که مخلوطی از دو روش بالاست . به کد زیر دقت کنید

```
class C : public B {
public:
    // ...
};
void f(int* p, int max)
{
    if (p) {
        // ...
```

```
    }
    for (int i = 0; i < max; ++i) {
        // ...
    }
}
```

چه زمانی براکت نیاز است ؟
تمام عبارات for , while , if , ... باید براکت داشته باشند چه چند خطی چه یک خطی ، این باعث می‌شود که وقتی بعداً کسی کد را مرور می‌کند و یا جمله‌ای را به عبارت اضافه می‌کند براکت آن را فراموش نکند هرچند که نگذاشتن براکت برای عبارات یک خطی اشکال سینتکسی ندارد ولی در استارندارد کد نویسی باید حتماً گذاشته شود

فضای خالی

این قسمت از کار همیشه به گروهی که در آن کار می‌کنید بستگی دارد .

1. از تب استفاده نکنید

استفاده از تب موجب می‌شود اگر تنظیمات تب شما با دیگران متفاوت باشد خوانایی کد پایین بیاید .
شما میتوانید از برنامه‌ای به اسم Artistic Style برای تبدیل تب به اسپیس استفاده کنید.آرتیستیک استایل با Cygwin نصب می‌شود و شما میتوانید با نوشتن در خط فرمان به آن دسترسی پیدا کنید . برای مک Mac App Store را ببینید تا کد خود را برای دندانه ها و تب ها تعمیر کنید قبل از ارسال از دستور استفاده کنید :

```
astyle -A2 -s4 myfile.cpp
```

اگر دوست دارید براکت‌های جدید را در خط جدید باز کنید از دستور زیر استفاده کنید :

```
astyle -A1 -s4 myfile.cpp
```

برای اطلاعات بیشتر صفحه astyle manpage را ببینید و میتوانید بای دریافت اطلاعات بیشتر از دستور زیر استفاده کنید :

```
astyle -h
```

نکته : ارتستیک استایل تب‌ها را در برنامه شما از بین میبرد نه در کامنت‌ها بنابراین شما نیاز دارید مطمئن شوید تب در کامنت های شما وجود ندارد .
راه ساده‌تر این است که از تنظیمات تکست ادیتور خود جایگزین تب را به عنوان 4 اسپیس انتخاب کنید

2. طول خطوط

همیشه طول خط خود را بر روی 80 کاراکتر تنظیم کنید چون میزان بیشتر خواند کد را در صفحات کوچکتر سخت می‌کند .

3. اسپیس اطراف اوپریتورها (+ ، - ، * و...)

همیشه قبل و بعد از اوپریتورهای باینری اسپیس بگذارید مخصوصاً >> و << این باعث افزایش خوانایی کد می‌شود مانند مثال زیر :

```
cin >> a;
double c = -a * b + 42 - d / 100;
cout << c << endl;
```

4. دندانه‌ها

همیشه با براکت ها دندانه گذاری کنید . از 3 یا 4 اسپیس برای این کار استفاده کنید . مثال :

```
int func() {
  int dragons = 0;
  cout << "Enter the number of dragons: ";
  cin >> dragons;
  while (dragons > 0) {
    Dragon d = getDragon();
    if (d.isFriendly()) {
      rideDragon();
    }
    dragons = dragons - 1;
  }
  return 42;
}
```

5. ایف و الس های پشت سر هم

همیشه else را با if آن تنظیم کنید . دو روش برای انجام این کار وجود دارد :

- قرار دادن else در همان خطی که if تمام می‌شود و قرار دادن if بعدی در همان خط و همینطور قرار دادن براکت آغازین در خطی که if و یا else وجود دارند :

```
if (condition) {          // Comment
                ...
        } else if (condition) {         // Comment
                ...
        } else {                        // Comment
                ...
```

```
    }
```

- قرار دادن براکت در خط بعدی و قرار دادن else if در یک خط :

```
if (condition)          // Comment
    {
        ...
    }
    else if (condition)      // Comment
    {
        ...
    }
    else                  // Comment
    {
        ...
    }
```

نه به اعداد جادویی

اعداد جادویی اعداد ثابتی هستند که بدون تعریف آن‌ها به عنوان ثابت از آن‌ها استفاده می‌کنید و این اعداد جادویی هستند چون بعد از 3 ماه از زده شدن کد هیچ‌کس بخاطر نمیآورد آن اعداد برای چه کاری بودند . به طور گسترده ای از اعداد 1 و 0 و -1 و 2 در کدها استفاده می‌شود که تعریف نکردن آن‌ها به عنوان ثابت ممکن است مشکل‌ساز شود مثال :

```
const int MY_CONSTANT = 10;
```

بخاطر معنی خاص نامشان آن‌ها را تماماً با حروف بزرگ بنویسید و از آندرلاین (_) به عنوان جدا کننده استفاده کنید .

نه به متغیرهای سراسری

یک متغیر سراسری تغیری است که بیرون هرگدونه تابع و کلاسی تعریف می‌شود و توسط همه توابع در دسترس است برای مثال در کد پایین myGlobal یک متغیر سراسری است .

```
#include <iostream>
using namespace std;

double myGlobal = 0;

int main(void) {
  // main function code
```

```
  myGlobal = 1;
}

void anotherFunc() {
  myGlobal = 2;
}
// other functions
```

به ندرت لازم می‌شود از متغیر سراسری استفاده کنید . این نوع تعریف متغیر خوانایی کد را به شدت کاهش میدهد . بنابر این بهتر از تا جای ممکن از آن‌ها استفاده نکنید .
در پایان بنده ، بهنام صباغی ، به عنوان مترجم و صد البته برنامه نویس زبان C++ به تمامی دوستان اعلام میکنم تا به اینجای مقاله ترجمه کاملاً از یک سایت خارجی به این آدرس انجام شده ولی در انتها لازم میدانم اعلام کنم با بیشتر موارد این مقاله موافقم بجز موردی که گفته شد بجای تب از اسپیس استفاده کنید و دلایلم را در زیر مینویسم تا اگر دوستان با من هم عقیده بودند از تب بجای اسپیس استفاده کنند :

1. منطقی‌تر این است بجای 4 اسپیس یا 8 اسپیس (وابسته به مدل کد زدن) از یک تب استفاده کنیم
2. استفاده از تب باعث می‌شود در مرور کد در ادیتور خود بسیار راحت‌تر باشیم مثلاً اگر در جایی 4 تب پست سر هم استفاده شده باشد و این کد را به طوری بنویسیم که بجای تب از اسپیس استفاده کنید برای رفتن از اول خط به آخر خط باید بجای 4 بار فشار دادن فلش رو به جلو 4*4 بار یعنی 16 بار فلش رو به جلو را فشار دهیم که اصلاً منطقی نیست
3. شاید در پروژه های کوچک مهم نباشد ولی وقتی میزان کد زیاد می‌شود با توجه به میزان زیادی فضای خالی که در کد وجود دارد استفاده از 4 کاراکتر اسپیس بجای یک کاراکتر تب میتواند حجم کد را به طور قابل توجهی افزایش دهد و این مسأله برای کسانی که با ورژن کنترل‌ها یا به صورت گروهی کار می‌کنند بسیار مهم است .
4. پروژه های بزرگی که این‌جانب بررسی کردم تماماً از تب استفاده می‌کنند برای مثال میتوانید به این کلاس از کد موتور قدرتمند کرای انجین مراجعه کنید و همینطور تمامی پروژه های اوپن سورس بزرگ دیگر نیز از تب بجای اسپیس استفاده می‌کنند .

موفق و پیروز باشید

بهنام صباغی