استاندارد برنامه‌نویسی
به زبان C++

C++ STYLE GUIDE

قاسم فودازی، محمد مطیعیان

دی ۱۳۹۲

بسم الله الرحمن الرحیم

فهرست

مقدمه ۱
۱-۱- شرح مختصر ۱
۱-۲- خلاصه‌ای از این مستند ۲
۱-۳- دسته‌بندی اولویت ۳
۲- سازماندهی فایلهای یک پروژه (File Organization) ۴
۲-۱- انواع فایل استفاده شده در زبان برنامه‌نویسی C++ ۴
۲-۲- دسته‌بندی فایل‌ها درون یک شاخه (Folder) ۵
۲-۳- سازماندهی درون فایل Header ۶
۲-۴- سازماندهی درون فایل CPP و C ۸
۲-۵- استفاده از فایل‌ها و کلاس‌های مشترک (Common and Global) ۹
۲-۶- استفاده از فایل‌های داده (Initialize, Config, temp, database, ...) ۱۰
۳- نام‌گذاری (Naming and Define) ۱۱
۳-۱- نام‌گذاری متغیرها ۱۱
۳-۲- نام‌گذاری فایل (Header and Cpp and C) ۱۲
۳-۳- نام‌گذاری کلاس (Class) ۱۲
۳-۴- نام‌گذاری استراکچرها (Structure) ۱۳
۳-۵- نام‌گذاری تابع و پارامترهای ورودی و خروجی آن (Function) ۱۳
۳-۶- تعریف شمارندهها (Enum) ۱۴
۳-۷- تعریف‌نوع (Typedef) ۱۵
۳-۸- تعریف‌ثابت (Const, Define, Macro) ۱۵
۴- کامنت‌گذاری (Comments) ۱۷
۴-۱- کامنت یک پروژه (فایل ReadMe) ۱۷
۴-۲- کامنت هدر فایل (Comments in Header) ۱۷
۴-۳- کامنت یک بلوک کد (Block Comments) ۱۸
۴-۴- کامنت یک خط کد (Single-Line Comments) ۱۸
۵- قواعد کلی کدنویسی ۱۹
۵-۱- رعایت انواع دسترسی و مخفی‌سازی در کلاس و استراکچر ۱۹
۵-۲- مدیریت حافظه (New & Delete) ۱۹
۵-۳- مقداردهی اولیه تمام متغیرها (Variable Initialization) ۲۰
۵-۴- عملگر مساوی یا مقدار دهی چندگانه (Multiple Assignments) ۲۰
۵-۵- حلقه‌ها (for & while & do) ۲۱
۵-۶- عبارت شرطی (if & switch) ۲۱

۷-۵- متغیرهای پایه و typdef (bool, char, int, float, double, …) ۲۲
۸-۵- فاصله‌گذاری (Using White Space and Tab) ۲۲

مقدمه

هدف این مستند ایجاد قوانینی می‌باشد تا، بتوان براساس آن نرم‌افزارهائی را به زبان C++ بصورت یکپارچه و با یک فرمت تولید کرد. همچنین معیاری برای ارزیابی نرم‌افزارهای تولید شده، باشد. امید است با رعایت اصول برنامه نویسی بتوان نرم‌افزارهائی با کیفیت و کارآمد تولید کرد و از همفکری و هم‌اندیشی همدیگر استفاده کرد.

۱-۱- شرح مختصر

این مستند، شامل اصول برنامه‌نویسی به زبان C++ می‌باشد. در حقیقت قوانین (Style Guide) برنامه‌های نرم‌افزاری به زبان C++ را مشخص می‌کند. در این مستند سعی شده‌است تا خوانائی، کیفیت و کنترل کدهای تولید شده را با یکسان‌سازی کدهای نرم‌افزاری C++ افزایش داد. همچنین مباحث شیءگرائی را در کدهای تولید شده اجرا کرد تا بتوان پروژه‌های بزرگ مشترک را در زمان مناسب و بهینه با تعداد افراد تیم بیشتر، تولید کرد.

یکی دیگر از مباحث مطرح شده، قوانین تجویزی می‌باشد، تا از بروز خطاهای رایج جلوگیری کند. این قوانین کمک می‌کند تا زمان کمتری صرف دباگ‌کردن و رفع خطاهای نرم‌افزارهای تولید شده، گردد.

۲-۱- خلاصه‌ای از این مستند

در فصل اول خلاصه‌ای از مستند ارائه می‌شود. همچنین دسته‌بندی اولویت اجرائی قوانین ذکر می‌شود.

فصل دوم به سازماندهی فایل‌های یک پروژه می‌پردازد. سعی شده‌است، اصولی برای تعریف و بکارگیری فایل‌های یک پروژه بیان شود. همچنین نحوه استفاده از فایل‌های مشترک و عمومی ذکر می‌شود.

در فصل سوم قوانین نامگذاری فایل‌ها، متغیرها، کلاس‌ها، توابع و ... بیان می‌شود. در این فصل سعی می‌شود، برای نام‌گذاری از دستورالعمل‌های مشخصی استفاده‌گردد تا خوانائی کدهای تولید شده، افزایش یابد.

فصل چهارم در مورد کامنت‌گذاری (Comments) در کدهای تولید شده، می‌باشد. این قوانین الزامات کامنت‌گذاری برای فایل‌ها، متغیرها و توابع می‌باشد.

در فصل پنجم قواعد کلی کدنویسی بیان می‌شود. در این فصل سعی می‌شود، یک‌سری قوانینی ذکر شود تا نحوه استفاده از دسترسی‌ها، مقداردهی اولیه، عمگرها، توابع عمومی، ... مشخص گردد.

در فصل ششم کلمات کلیدی بیان می‌شود. این کلمات کلیدی شامل حلقه‌ها، عبارات شرطی، متغیرهای پایه می‌باشد.

فصل هفتم برای جمع‌بندی و نکات کلیدی می‌باشد.

۳-۱- دسته‌بندی اولویت

در این مستند برای بیان اهمیت و اولویت اجرائی قوانین و قواعد ذکر شده، از چهار اولویت استفاده می‌شود. این چهار اولویت عبارت‌است از:

اجباری و ضروری (Mandatory) : علامت این الزام **[Man]** می‌باشد. به این معنا، که عبارت ذکر شده، لازم‌الاجراست و در کد نوشته شده، حتما باید اجرا شود. معمولا در عبارت ذکر شده از کلمات "حتما، الزاما، اجبارا، ..." استفاده می‌شود.

جایگزین (Alternative) : علامت این الزام **[Alt]** می‌باشد. به این معنا، که یکی از موارد عبارت ذکر شده، لازم‌الاجراست و در کد نوشته شده، حتما باید اجرا شود. در حقیقت به کدنویس این اختیار داده می‌شود که یکی از موارد ذکر شده را انتخاب کند. در جملات و عبارات ذکر شده معمولا از کلمات "بجای، جایگزین، ..." استفاده می‌شود.

اختیاری (Optional) : علامت این الزام **[Opt]** می‌باشد. به این معنا، که عبارت ذکر شده، اختیاری است. این اولویت برای بهبود خوانائی و کارائی کد بیان شده‌است. در جملات و عبارات ذکر شده معمولا از کلمات "اختیاری، می‌توان، مناسب است، ..." استفاده می‌شود.

پیشنهادی (Recommended) : علامت این الزام **[Rec]** می‌باشد. به این معنا، که عبارت ذکر شده، در حال‌حاضر یک پیشنهاد می‌باشد. در صورت صلاحدید در آینده تبدیل به اجباری یا جایگزین می‌گردد. در جملات و عبارات ذکر شده معمولا از کلمات "پیشنهادی، بهتراست، ..." استفاده می‌شود.

این علامات در آخر جملات و یا پاراگراف داخل براکت ([]) قبل از نقطه انتهای جمله آورده می‌شود. برای نمونه عبارت زیر اولویت اجباری دارد:

"در ابتدای فایل هدر (Header File) حتما کامنت در مورد فایل باشد**[Man]**.

۲-سازماندهی فایل‌های یک پروژه (File Organization)

در این بخش سعی می‌شود، فایل‌های یک پروژه بخوبی مدیریت گردند. در بخش ۱-۲- ابتدا به نحوه استفاده از انواع فایل‌ها درون یک پروژه می‌پردازیم. سپس در بخش ۲-۲- برای ایجاد نظم، به دسته‌بندی فایل‌ها درون شاخه‌های مجزا می‌پردازیم. در بخش‌های ۳-۲- و ۴-۲- به سازمان‌دهی درون فایل‌های CPP و H پرداخته می‌شود. در بخش ۵-۲- چگونگی استفاده از فایل‌های ایجاد شده مشترک و عمومی مشخص می‌کند. در بخش ۶-۲- بطورخاص چگونگی استفاده از فایل‌های Config، Initialize، Temp، ... که برای نگهداری داده‌ها اولیه و اطلاعات تنظیمات و داده‌های لوگ‌گیری را مشخص می‌کند.

۱-۲- انواع فایل استفاده شده در زبان برنامه‌نویسی C++

فایل‌های متنوعی در یک پروژه استفاده می‌گردد. بر حسب بخش پسوند فایل می‌توان این نوع فایل‌ها را دسته‌بندی کرد. در ادامه به معرفی و بررسی این نو فایل‌ها می‌پردازیم.

فایل CPP: این فایل معمولا حاوی کدهای اجرائی زبان C++ می‌باشد. درون این فایل می‌تواند شامل کدهائی به زبان C و یا ASM باشد. این نوع فایل توسط کامپایلر تحلیل می‌شود. به ازای هر فایل CPP یک فایل هدر (فایل H) باید باشد**[Man]**. در بخش ۴-۲-به سازماندهی درون این نوع فایل می پردازیم.

فایل C: این فایل شامل کدهای اجرائی زبان C می‌باشد. تا حد امکان از این نوع فایل و زبان C استفاده نگردد**[Alt]**. مگراینکه تبدیل کدهای آماده‌ای که (شبیه کدهای گرفته شده از شرکت‌های دیگر) زمانبر و دشوار باشد**[Alt]**. در صورت نیاز به نوشتن کدهای به زبان C از دستور زیر استفاده گردد.

```
#ifdef __cplusplus
extern "C" {
#endif
```

سپس کد به زبان C نگارش شود. و در انتهای کد این خطوط اضافه گردد.

```
#ifdef __cplusplus
```

```
}
#endif
```

فایل H: این فایل معمولا حاوی تعاریف و قالب‌های کد به زبان C++ می‌باشد. این نوع فایل توسط کامپایلر تحلیل نمی‌شود مگراینکه توسط فایل CPP فراخوانی (include) شده باشد. در صورت فراخوانی فایلH درون فایل CPP، عملکرد کامپایلر شبیه به این می‌باشد که، کل فایل H را در همان نقطه فراخوانی به کد CPP اضافه می‌کند تا تحلیل شود. به همین‌دلیل استفاده از فراخوانی‌های متعدد و تودرتو منجر به خطا می‌شود.

فایل LIB: این نوع فایل‌ها شامل بخش کامپایل‌شده و آماده کد زبان CPP می‌باشند. این فایل‌ها سرعت کامپایل‌کردن پروژه را افزایش می‌دهند. این نوع فایل‌ها معمولا با یک فایل H توصیف می‌شوند. فایل‌های LIB حتما کنار فایل H خودش قرار بگیرد[Man].

فایل DLL: این فایل‌ها بمانند تکه‌ای از کد در بیرون از پروژه می‌باشند. خروجی بعضی از پروژه‌ها از این نوع فایل می‌باشد. برای استفاده از این فایل‌ها در پروژه یک فایل LIB برای برقراری لینک با DLL و یک فایل H برای توصیف توابع و کلاس درون DLL نیاز است. فایل DLL باید درون شاخه اجرایی (شاخه‌های Debug و Release) قرار گیرد[Man]. فایل DLL موقع اجرای فایل EXE پروژه، در حافظه بارگذاری می‌گردند.

فایل EXE: تقریبا خروجی همه پروژه‌ها از این نوع فایل می‌باشد. این فایل باید درون شاخه اجرایی (شاخه‌های Debug و Release) قرار گیرد[Man]. فایل EXE بعنوان محصول پروژه می‌باشد. محصول نهائی یک پروژه باید در نسخه Release تحویل داده شود[Man].

۲-۲- دسته‌بندی فایل‌ها درون یک شاخه (Folder)

فایل‌های یک پروژه را براساس کاربری و عملکردشان درون شاخه‌های مختلف قرار می‌دهیم. پروژه باید درون شاخه‌ای به همان اسم پروژه قرار گیرد[Man].

دو شاخه Debug و Release برای خروجی پروژه در نظر گرفته شده‌است. تمام فایل‌ها و شاخه‌هایی موردنیاز یک پروژه برای اجرا باید درون این دو شاخه Debug و Release باشند[Man]. شاخه Debug برای

نگهداری نوعی از خروجی پروژه می‌باشد که معمولا با سربار باگ‌گیری بوده و برای رفع باگ و دباگ‌کردن و تست نرم‌افزار از آن استفاده می‌شود[Rec]. شاخه Release برای نگهداری نوعی از خروجی پروژه می باشد که معمولا با بدون سربار باگ‌گیری بوده و خروجی نهائی نرم‌افزار می‌باشد و برای اجرا بهینه‌سازی شده‌اند[Rec].

فایل‌های از نوع Temp و Log که توسط نرم‌افزار بطور اتوماتیک تولید می‌گردند، باید درون شاخه Temp و یا Log قرار گیرند[Man]. بهتراست مکانیسمی برای پاک‌کردن فایل‌های Temp و Log قدیمی در پروژه تعبیه گردد تا مشکل محدودیت سخت‌افزاری پیش نیاید[Rec].

فایل‌های نوع داده اولیه و تنظیمات (Initilize، Config، Data، ...) باید درون شاخه Config و یا Init و یا Data قرار گیرند[Man].

فایل‌هائی که با هم یک عملیات خاص را انجام می‌دهند (شامل CPP، C، H، LIB) باید درون یک شاخه به اسم همان عملیات قرار گیرند[Man]. برای مثال دسته فایل‌هائی که وظیفه دیکدکردن یک استاندارد را دارند.

فایل‌های مشترک و عمومی (Common & Global) حتما درون شاخه خودشان و در بیرون پروژه قرار گیرند[Man]. بهتر است این فایل‌ها بصورت Read Only باشند تا جلوی ویرایش آن‌ها در غیر مواقع ضروری گرفته شود.

۳-۲- سازماندهی درون فایل Header

فایل‌های H جزو اصلی‌ترین فایل‌های یک پروژه می‌باشند. این فایل‌ها توصیف‌کننده نحوه عملکرد یک پروژه می‌باشند. به همین‌خاطر قالب استاندارد سخت‌گیرانه‌ای برای آن‌ها در نظر گرفته‌می‌شود. درون این فایل‌ها از بالا به پائین باید براساس استراکچر زیر تعریف شوند[Man].

ابتدای فایل باید با توضیحات یا کامنت‌گذاری مربوطه مشخص شود[Man]. قالب این کامنت‌گذاری در بخش ۲-۴- مشخص شده‌است.

پس‌از کامنت‌گذاری لازم است چند خط تعریف زیر برای کامپایلر صورت گیرد. این خطوط بمنظور جلوگیری از دوباره کامپایل شدن فایل در یک مرحله کامپایل می‌باشد[Alt]. البته در کامپایلرهائی که Wizard اتوماتیک آن‌ها تنها #pragma once در ابتدای فایل قرار می‌دهند، همان کفایت می‌کند[Alt]. بهتر است برای همخوانی در کامپایلرها این خطوط تعریف اولیه قرار گیرند، مخصوصا برای فایل‌هائی که وظیفه پردازش را دارند و از نوع دیالوگ و فریم نمی‌باشند[Rec].

```
#if !defined(_#NameOfFile_H_)
#define _#NameOfFile _H_
#if _MSC_VER > 1000
#pragma once
#endif // _MSC_VER > 1000
```

البته بجای #NameOfFile باید اسم فایل با حروف بزرگ آورده شود. (برای تعریف _#NameOfFile_H_ به بخش ۸-۳- مراجعه کنید.) البته در انتهای فایل H هم برای بستن این تعریف از خط زیر استفاده گردد[Man].

```
#endif //!defined _#NameOfFile _H_
```

سپس باید خطوط تعریف اضافه‌کردن (include) فایل‌های H گنجانده شود[Rec]. البته باید سعی شود که فقط Hفایل‌های ضروری اضافه گردد. بهتر است تا حد امکان include فایل‌ها به درون فایل C و CPP برده شوند[Rec].

بعداز آن بترتیب این بخش‌های کد نگارش شود: ماکروها، typdefها، متغیرهای عمومی، توابع عمومی، استراکچرها، کلاس‌ها[Alt]. البته در مواردی که احتمال مشکل تقدم و تاخر برای کامپایل وجود دارد، می‌توان در این ترتیب تغییر ایجاد کرد[Alt].

درون استراکچرها و کلاس‌ها، ابتدا متغیرهای اصلی (قابل دسترسی از بیرون کلاس و استراکچر) در بخش Public آورده شود[Rec]. سپس توابع مولد Constructors و تابع مخرب Deconstructors آورده شود. بعد از آن توابع عملگر مساوی و دیگر عملگرها نگارش شود. بعد از آن توابع اصلی کلاس و استراکچر باشد. سپس در بخش Private ابتدا متغیرهای خاص و سپس توابع خاص آورده‌شود[Rec]. در مورد کلاس‌هائی که

اتوماتیک توسط Wizard تولید می‌شوند، بخش‌های اصلی تولید شده، توسط Wizard به‌همان سبک باقی بماند[Rec].

بهتر است بجای استراکچر از کلاس استفاده شود**[Opt]**. سعی شود در مواقعی از استراکچر استفاده گردد که حاوی ساختمانی از داده باشد و توابع مربوطه در استراکچر فقط در تولید، ویرایش و حذف آن داده‌ها فعال باشند. یعنی استراکچر عملیات‌های اجرائی، دستوری و ارتباطی را پشتیبانی نکند. بهتر است تعریف استراکچرهائی که فقط درون یک کلاس استفاده می‌شوند را، درون فایل H بعنوان عضو کلاس تعریف شوند**[Opt]**.

بهتر است بجای union از استراکچر استفاده گردد**[Opt]**. سعی شود در مواقعی از union استفاده گردد که نیاز به استراکچر چندگانه برای تبادل وجایگشت داده باشد. و حتما استراکچرهای داخل union از نظر اندازه حافظه برابر باشند، و بجای حافظه خالی از متغیر Spare استفاده گردد[Rec].

متغیرهای عمومی و مقادیر ثابت حتما بعنوان عضو کلاس بصورت static const و یا enum تعریف شوند[Rec].

درون فایل H از تعریف محتوای داخلی توابع کلاس و یا استراکچر، پرهیز شود[Alt]. مگر در مواردی که استراکچر عضو کلاس است و تعریف این استراکچر درون کلاس، طولانی نباشد[Alt].

۴-۲- سازماندهی درون فایل CPP و C

فایل‌های C و CPP بعنوان نگهدارنده کد عملیاتی فایل‌های H می‌باشند. در هنگام کامپایل این فایل‌های C و CPP هستند که توسط کامپایلر مورد ارزیابی، تحلیل و تبدیل به اسمبلی می‌گردند. درون این فایل‌ها از بالا به پائین باید براساس ساختار زیر تعریف شوند.

ابتدای فایل می‌توان کامنت‌گذاری در مورد عملکرد فایل نوشت**[Opt]**.

سپس باید خطوط تعریف اضافه‌کردن include فایل‌های H گنجانده شود. البته ابتدا include فایل stdafx.h قرار گیرد. سپس include فایل H مربوط به CPP قرار گیرد**[Man]**. توجه شود هر فایل C و CPP

حتما باید یک فایل H همنام داشته باشد که، تعریف قالب‌ها و ساختارهای فایل C و CPP را بیان کند**[Man]**.

پس از آن Hفایل‌های موردنیاز اضافه گردد.

بعد از خطوط include فایل‌ها متغیرهای عمومی و static و سپس توابع عمومی تعریف گردند[Rec].

سپس کد درون توابع کلاس‌ها به ترتیبی که درون فایل H تعریف می‌شوند، نگارش شود. سعی شود بین تعریف توابع یک خط فاصله قرار گیرد.

درون توابع کلاس و یا عمومی سعی شود که، کد درون تابع بصورت بلوکی نگارش شود. و وظیفه هر بلوک در توضیحات خلاصه‌ای اول بلوک مشخص شود. باید در ابتدای تابع، تعریف متغیرهای محلی موردنیاز تابع قرار گیرد. البته می توان در هر بلوک متغیر موردنیاز آن بلوک را تعریف کرد.

۵–۲- استفاده از فایل‌ها و کلاس‌های مشترک (Common and Global)

فایل‌ها و کلاس‌های مشترک معمولا بعلت پرکاربرد بودن و تکرار استفاده آن‌ها در پروژه‌های متعدد، بعنوان یک Object و شئ فراگیر می‌باشند. قوانین استاندارد C++ Style Guide باید بصورت کامل برای این دسته از فایل‌ها اجرا شود.

بهتر است که فایل‌ها و کلاس‌های مشترک بصورت فایل DLL تبدیل شوند تا از DLL آن‌ها در پروژه استفاده گردد. تاریخ ویرایش و ورژن فایل DLL حتما درج گردد**[Opt]**.

بهتر است که فایل‌ها و کلاس‌های مشترک درون شاخه Common در بیرون پروژه قرار گیرند[Rec].

به هیچ وجه فایل‌های (Common & Global) دستکاری و شخصی‌سازی نگردند**[Man]**. در صورت‌نیاز کپی از آن‌ها درون پروژه شخصی‌سازی گردد.

ساختار این فایل‌ها باید بگونه‌ای باشد که درصورت بروز شدن، توابع و عملکرد جدیدی به آن اضافه گردد و تمام توابع و عملکردهای قبلی را هم پوشش دهد[Rec]. تا در اضافه‌کردن آن‌ها به پروژه‌ها و بروز شدن آن‌ها مشکلات عدم تطبیق کدها پیش نیاید.

۶-۲- استفاده از فایل‌های داده (Initialize, Config, temp, database, ...)

فایل‌های داده شامل Initialize، Config، temp، database، ... حاوی یک سری اطلاعاتی می‌باشند که برای کارکرد نرم‌افزار با تنظیمات خاص یا دستی و همچنین اطلاعات اولیه پردازش‌های درون کد و لوگ‌گیری از فرایند اجرای نرم‌افزار می‌باشند. این دسته از فایل‌ها، همراه فایل اجرائی باید باشند تا طبق فرایند موردنیاز کاربر، نرم‌افزار عمل کند. این فایل‌ها به سه دسته تقسیم می‌شوند.

دسته اول فایل‌های Initialize و Config می‌باشند. این فایل‌ها باید با پسوند ini و یا cfg و یا هر پسوندی با نوع فایل خواندنی مانند txt یا XML باشند. حتما این فایل‌ها را درون شاخه با نام ini و یا config قرار گیرند[Alt]. در صورتی‌که فقط یک فایل از این نوع همراه نرم‌افزار باشد، می‌توان در شاخه اصلی نرم‌افزار قرار داد[Alt].

دسته دوم فایل‌های database و data می‌باشند. این فایل‌ها بعنوان داده‌های برنامه می‌باشند که از نوع ورودی و یا خروجی نرم‌افزار تلقی می‌گردند. این فایل‌ها دارای پسوندهای متنوعی می‌باشند. بهتر است این فایل‌ها براساس محتوائی که دارند، دسته‌بندی شده، درون شاخه‌هائی با نام دسته‌شان قرار گیرند[Opt]. از بهمراه داشتن و یا تولید این فایل‌ها توسط نرم‌افزار در شاخه اصلی به تعداد بیش‌از سه فایل پرهیز شود[Opt].

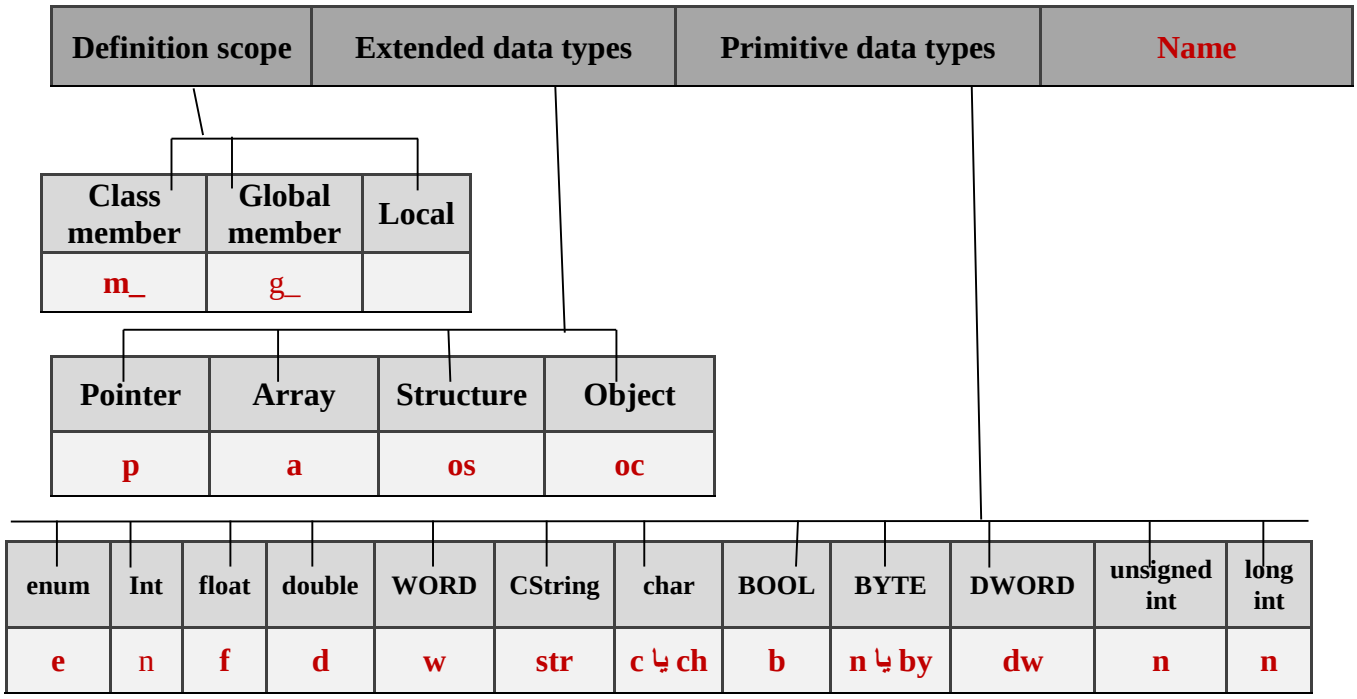
دسته سوم فایل‌های temp و log می‌باشند. این فایل‌ها معمولا برای رصد عملکرد نرم‌افزار هنگام اجرا تولید می‌شوند. پسوند این فایل‌ها بهتر است از نوع tmp و یا log باشند. این فایل‌ها حتما درون شاخه با نام tmp و یا temp و یا log قرار گیرند[Man].

۳-نام‌گذاری (Naming and Define)

استاندارد نام‌گذاری در هر زبان برنامه‌نویسی باید به صورت گویا باشد، تا بیان‌کننده و توصیف‌کننده کد نگارش شده، باشد. در ادامه اصول کلی برای نام‌گذاری متغیرها، کلاس‌ها، استراکچرها، فایل‌ها و ... قواعدی بیان می‌گردد تا خوانائی کدهای تولید شده بزبان C++ بهتر گردد.

۳-۱- نام‌گذاری متغیرها

برای نام‌گذاری یک متغیر چند بخش در نام متغیر مدنظر می‌باشد. در جدول زیر این بخش‌ها آمده‌است[Man].



در قسمت مربوط به Extended Data Types اولویت از چپ به راست می باشد برای نمونه اگر یک متغیر عضو کلاس اشاره گری به آرایه از نوع کاراکتر داشته باشیم، نام متغیر را به صورت زیر می‌نویسیم:

m_ pachName

برای تعریف یک متغیر از نوع آرایه‌ای از نوع کاراکتر و به صورت محلی می‌توان به صورت زیر نام متغیر را تعریف کرد:

achName

نام متغیر باید برگرفته از عملکرد متغیر باشد و می‌تواند شامل چند کلمه باشد. برای خوانائی نام متغیر همه حروف آن کوچک می‌باشد، بجز حروف اول کلمات که حروف بزرگ می‌باشند[Man].

این نام‌گذاری برای متغیرهای کلاس و گلوبال اجباری است[Man]. ولی برای متغیرهای محلی توابع اختیاری می‌باشد[Opt].

۲-۳- نام‌گذاری فایل (Header and Cpp and C)

نام فایل باید برگرفته از عملکرد محتویات آن باشد و می‌تواند شامل چند کلمه باشد. برای خوانائی نام فایل همه حروف آن کوچک می‌باشد، بجز حروف اول کلمات که حروف بزرگ می‌باشند[Man]. توجه شود که نام فایل‌ها در صورت داشتن کلاس اصلی، هم‌نام کلاس اصلی باشد.

۳-۳- نام‌گذاری کلاس (Class)

نام کلاس‌ها با حرف C (بزرگ) شروع می‌شوند. برای هر کلاس دو فایل CPP و H تولید می‌شود. نام کلاس باید برگرفته از عملکرد کلاس باشد و می‌تواند شامل چند کلمه باشد. برای خوانائی نام کلاس همه حروف آن کوچک می‌باشد، بجز حروف اول کلمات که حروف بزرگ می‌باشند[Man]. برای نام‌گذاری از حروف استفاده شود و استفاده از علائم دیگر پرهیز شود[Man]. برای مثال کلاسی که وظیفه Modulation Detection را برعهده دارد، به صورت زیر می‌نویسیم:

CModulationDetect

که برای آن ۲ فایل زیر تولید می‌شود:

ModulationDetect.cpp , ModulationDetect.h

اگر کلاس از کلاس دیگری ارث‌بری کند، بهتر است که در انتهای نام کلاس، نام کلاس ارث برده شده و یا مخفف نام آن آورده شود[Opt]. برای مثال اگر کلاس نمونه بالا از کلاس CDialog ارث بری کند، می‌توان نام کلاس را به صورت زیر بنویسیم:

CModulationDetectDlg

۴-۳- نام‌گذاری استراکچرها (Structure)

نام استراکچرها با حرف tag (کوچک) شروع می‌شوند. نام استراکچر باید برگرفته از عملکرد آن باشد و می‌تواند شامل چند کلمه باشد. برای خوانائی نام استراکچر همه حروف آن کوچک می‌باشد، بجز حروف اول کلمات که حروف بزرگ می‌باشند[Man]. برای نام‌گذاری از حروف استفاده شود و استفاده از علائم دیگر پرهیز شود[Man]. برای مثال استراکچر با عنوان User Name به صورت زیر می‌نویسیم:

tagUserName

اگر استراکچر از استراکچر دیگری ارث‌بری کند، بهتر است که در انتهای نام آن، نام استراکچر ارث برده شده و یا مخفف نام آن آورده شود[Opt]. برای مثال اگر استراکچر نمونه بالا از استراکچر tagPersonal ارث‌بری کند، می‌توان نام استراکچر را به صورت زیر بنویسیم:

tagUserNamePrs

۵-۳- نام‌گذاری تابع و پارامترهای ورودی و خروجی آن (Function)

نام تابع باید برگرفته از عملکرد آن باشد و می‌تواند شامل چند کلمه باشد. برای خوانائی نام تابع همه حروف آن کوچک می‌باشد، بجز حروف اول کلمات که حروف بزرگ می‌باشند[Man]. برای نام‌گذاری از حروف استفاده شود و استفاده از علائم دیگر پرهیز شود[Man]. برای مثال تابعی با عنوان Get Character به صورت زیر می‌نویسیم:

```
char  GetCharacter ()
```

تابع تا حد امکان خروجی داشته باشد (برای مثال خروجی از نوع BOOL) ولی می‌تواند بدون خروجی با عبارت void باشد. پارامترهای ورودی تابع، باید بترتیب از نوع ورودی و سپس پارامترهای خروجی باشد[Man]. تا حد امکان نوع پارامترهای ورودی تابع از لحاظ ورودی یا خروجی بودن مشخص گردد[Rec]. بعنوان مثال با پیشوند و یا پسوند In و Out مشخص گردد. نام‌گذاری پارامترهای ورودی تابع از قواعد بخش ۱-۳- پیروی می‌کند.

۶-۳- تعریف شمارنده‌ها (Enum)

نام شمارنده‌ها با حرف en و یا enum (کوچک) شروع می‌شوند. نام شمارنده باید برگرفته از عملکرد آن باشد و می‌تواند شامل چند کلمه باشد. برای خوانائی نام شمارنده همه حروف آن کوچک می‌باشد، بجز حروف اول کلمات که حروف بزرگ می‌باشند[Man]. برای نام‌گذاری از حروف استفاده شود و استفاده از علائم دیگر پرهیز شود[Man]. برای نام‌گذاری آیتم‌های درون شمارنده با حرف e (کوچک) شروع می‌شوند و نام‌گذاری آن‌ها از قواعد نام شمارنده تبعیت می‌کند. برای مثال شمارنده‌ای با عنوان User Access را به صورت زیر می‌نویسیم:

```
enum enUserAccess
{
        eUserAccessDemo  = 0,
        eUserAccessAdmin,
        eUserAccessOperator
};
```

بهتر است تا حدامکان شمارنده‌ها، عضو کلاس یا استراکچر باشند. مخصوصا شمارنده‌هائی که عملکردشان مختص یک کلاس می‌باشند[Opt].

۷-۳- تعریف‌نوع (Typedef)

نام تعریف‌نوع با حرف typ (کوچک) شروع می‌شوند. نام تعریف‌نوع باید برگرفته از عملکرد آن باشد و می‌تواند شامل چند کلمه باشد. برای خوانائی نام تعریف‌نوع همه حروف آن کوچک می‌باشد، بجز حروف اول کلمات که حروف بزرگ می‌باشند[Man]. برای نام‌گذاری از حروف استفاده شود و استفاده از علائم دیگر پرهیز شود[Man]. از تعریف‌نوع برای خلاصه‌کردن نام تعریف و یا ساده‌کردن جایگزینی type می‌استفاده شود.

۸-۳- تعریف‌ثابت (Const, Define, Macro)

نام تعریف‌ثابت شامل (Const, Define, Macro) با حروف بزرگ نوشته می‌شوند[Man]. نام تعریف‌ثابت باید برگرفته از عملکرد آن باشد و می‌تواند شامل چند کلمه باشد. برای خوانائی نام تعریف‌ثابت کلمات با علامت (_) از هم جدا می‌شوند[Man].

توجه شود که تعریف‌ثابت Const از نوع متغیر می‌باشد و برای نام‌گذاری آن هم می‌توان از روش نام‌گذاری متغیرها با پیشوند c استفاده کرد[Rec].

اگر تعریف‌ثابت مختص کلاس یا استراکچر نوشته شده‌است، بهتر عضو همان کلاس و از نوع Const تعریف شود[Rec].

تعریف‌ثابت Define که بعنوان IDعمکردی می‌باشد، باید در stdafx و یا گلوبال فایل و یا hفایل کلاس استفاده کننده، قرار گیرد[Man]. تعریف‌ثابت Define که بعنوان IDکنترلی می‌باشد، باید در Resource فایل قرار گیرد[Man]. برای خوانائی از (_) مابین کلمات اسم ID استفاده شود. در جدول زیر بسته به‌اینکه IDکنترلی و یا عملکردی از چه نوعی باشد، پیشوند آن مشخص شده‌است.

ID type	Example	توضیح
IDD_	IDD_DIALOG_CONNECTION IDD_DLG_CONNECTION IDD_FORM_CONNECTION IDD_CONNECTION	این پیشوند ID برای دیالوگ‌ها و فرم‌ها استفاده می‌شود. نام دیالوگ و فرم همنام اسم کلاس مربوطه می‌باشد.
IDB_	IDB_ANT IDB_BMP_ANT	این پیشوند ID برای تصاویر Bitmap استفاده می‌شود.
IDI_	IDI_ANT IDI_ICN_ANT	این پیشوند ID برای تصاویر Icon استفاده می‌شود.
IDS_	IDS_COUNTRY_NAME IDI_STR_ COUNTRY_NAME	این پیشوند ID برای جملات پیش‌فرض استفاده می‌شود.
IDC_	IDC_BUTTON_CONNCT IDC_BTN_CONNCT IDC_COMBO_CONNCT IDC_EDIT_CONNCT IDC_RADIO_CONNCT IDC_STATIC_CONNCT	این پیشوند ID برای IDهای کنترلی داخل res فایل استفاده می‌شود. بهتر است نام مخفف یا کامل نوع ID نوشته نیز شود.
IDR_	IDR_MENU_MAIN IDR_TLB_MAIN IDR_PNG_BOX	این پیشوند ID برای همه انواع IDکنترلی می‌توان استفاده کرد. البته بهتر است برای IDهای مشخص بالا از همان نگارش استفاده کرد. همچنین برای مشخص شدن نوع ID، نام مخفف یا کامل نوع ID نوشته شود.
ID_	ID_COMMAND_PLAY ID_DATA_SAVE ID_INIT_FIRST	این پیشوند ID برای همه انواع IDعملکردی می‌توان استفاده کرد. این IDها بطور عمومی در کد استفاده می‌شوند.
WM_	WM_COMMAND_PLAY WM_DBMANAGER WM_DATA_RECEIVE	این پیشوند ID برای همه انواع WMپارامتر می‌توان استفاده کرد. این WMها از نوع WM_USER در کد می‌باشند.

۴- کامنت‌گذاری (Comments)

کامنت‌گذاری بمنظور توضیح دادن همه یا بخشی از کد می‌باشد. همچنین کامنت‌گذاری باید خلاصه و گویا باشد. از کامنت‌گذاری نابجا، توضیحات اضافه، توضیحات ناقص و گمراه‌کننده پرهیز شود. در این فصل به کامنت‌گذاری در یک پروژه و زیربخش‌های آن می‌پردازیم.

۴–۱- کامنت یک پروژه (فایل ReadMe)

فایل ReadMe یکی از فایل‌های اصلی یک پروژه می‌باشد[Man]. این فایل گویای عملکرد پروژه می‌باشد. این فایل مستند معرفی پروژه نیز می‌باشد. کامنت‌گذاری فایل ReadMe سه بخش معرفی، نحوه‌عملکرد و نتیجه دارد. معمولا می‌توان بخش نحوه‌عملکرد را در کد نگارش یافته دید، و می‌توان آن‌را از ReadMe حذف کرد.

در بخش معرفی به کلیاتی درباره پروژه و علت وجودی آن پرداخته شود. همچنین مدیر و اعضای اصلی بوجودآورنده پروژه مشخص گردد. در بخش نحوه عملکرد، به کلیاتی درباره چگونگی عملکرد پروژه پرداخته شود. همچنین ورودی و خروجی‌های مهم پروژه ذکر شود. در بخش نتیجه، خروجی اصلی و کارکرد اصلی پروژه ذکر گردد.

۴–۲- کامنت هدر فایل (Comments in Header)

کامنت‌گذاری فایل Header بترتیب با توضیحات جدول زیر می‌باشد. بعد از توضیحات اجباری جدول زیر، می‌توان توضیحات دیگر موردنیاز را گنجاند.

Field	Example	توضیح
File Description	// String.h: support interface for the string in ANSI and UNIcode and ..	ابتدا اسم فایل مشخص می‌شود. و سپس خلاصه‌ای از عملکرد محتوای درون فایل در یک یا چند خط بیان می‌شود. سعی شود کلاس‌ها و توابع و متغیرهای اصلی در این خلاصه گنجانده شود.
Contenets	// Classes:CWString, CAString // Function:Memcopy, Memset	در این بخش باید اسامی کلاس‌ها و توابع عمومی (Global) مشخص شود.
Author	// Written by: Name.Family	نام یا نام‌های افراد پدیدآورنده آورده‌شود.
Date	// Date: 2013/09/10	تاریخ ایجاد اولیه فایل آورده شود. نوع و فرمت تاریخ اختیاریست.
Change History	<Author Date Description>	تاریخ ایجاد اولیه فایل آورده شود. نوع و فرمت تاریخ اختیاریست.

۳-۴- کامنت یک بلوک کد (Block Comments)

کامنت‌گذاری یک بلوک کد برای توصیف عملکرد یک بلوک از کد می‌باشد. شامل دو بخش نحوه‌عملکرد و نتیجه بلوک می‌باشد. در بخش نحوه‌عملکرد، پیچیدگی بلوک بسادگی توضیح داده شود. و در بخش نتیجه خروجی و هدف بلوک بیان شود. کامنت‌گذاری باید قبل از خط کد نوشته شود.

۴-۴- کامنت یک خط کد (Single-Line Comments)

کامنت‌گذاری یک خط کد مواقعی صورت گیرد که، بدلیل وجود استفاده از تابع با عملکرد نامشخص و یا طولانی شدن خط کد، گویائی و خوانائی کد تحت تاثیر قرار گیرد. بهتر است کامنت‌گذاری قبل از خط کد نوشته شود و یا روبروی خط کد نگارش شود.

۵- قواعد کلی کدنویسی

در این فصل سعی می‌شود موارد کلی که در فصول قبل به آن نپرداختیم، بیان شود. این قواعد برای خوانائی، تطبیق‌پذیری و کاهش خطاهای رایج کدنویسی می‌باشد.

۵-۱- رعایت انواع دسترسی و مخفی‌سازی در کلاس و استراکچر

متغیرها و توابع که خارج‌از کلاس به آن‌ها احتیاج داریم و بعنوان رابط کلاس و خارج از کلاس می‌باشند، در سطح دسترسی public قرار گیرند. مابقی متغیرها و توابع در سطح دسترسی private قرار گیرند. سطح دسترسی protected برای حفاظت متغیرها و توابع در هنگام ارث‌بری می‌باشد. برای استراکچر هم از این قواعد پیروی کنید. در بخش ۳-۲- به ترتیب قرار گرفتن دسترسی‌ها درون کلاس و استراکچر پرداختیم.

۵-۲- مدیریت حافظه (New & Delete)

بطورکلی متغیری که ایجاد و تولید (new) شده‌است، باید با مکانیسم مناسبی حذف و پاک‌سازی (delete) گردد. این وظیفه برعهده محل ایجاد کننده متغییر (یعنی تابع یا کلاس و یا استراکچر) می‌باشد. مگر اینکه با مکانیسم مناسبی به تابع یا کلاس و یا استراکچر دیگری تنفیذ شود. همانند ایجاد متغیر در یک تابع و ارسال (PostMessage) آن به تابع کلاس دیگر و حذف آن در تابع کلاس دوم.

توجه شود که متغیر آرایه‌ای که با [] new ایجاد شده، حتما با [] delete حذف گردد**[Man]**.

اگر متغیری بصورت تعریف درجا ساخته شود از حافظه stack استفاده می‌کند و اگر بصورت تعریف new ایجاد شود، از حافظه سیستم (RAM و یا Virtual RAM) استفاده می‌کند. تعریف آرایه‌های متغیری با اندازه حافظه بزرگ‌تر از 10Kbyte را حتما با new صورت گیرد. مگر اینکه این متغیر عضو کلاس و استراکچری می‌باشد، که حتما آن کلاس با new ایجاد می‌شود.

بهتر است که متغیرهای آرایه‌ای درون کلاس و استراکچر داخل تابع مولد کلاس (Construction) و یا تابع مخصوص ایجاد و initialize تولید شوند. سپس در تابع مخرب (Deconstruction) این آرایه متغیری حذف و پاک‌سازی شود.

این ایجاد و حذف برای Handleها هم صادق است. نحوه ایجاد Handle با صدا زدن توابعی همانند Create و Open می‌باشد و پاک‌سازی آن Handle با Delete و Close صورت می‌گیرد. برای مثال Handle فایل با تابع Open ایجاد می‌شود و با تابع Close پاک‌سازی می‌شود. و یا Handle یک Object با تابع Create تولید می‌شود و با تابع Delete پاک‌سازی می‌گردد.

۳–۵- مقداردهی اولیه تمام متغیرها (Variable Initialization)

متغیرهای یک کلاس و یا استراکچر باید در تابع مولد کلاس (Construction) مقداردهی اولیه مناسب گردند[Alt]. مگر اینکه مقدار اولیه متغیر اهمیت نداشته باشد و احتیاج به ایجاد آرایه بزرگی از این متغیر باشد[Alt]. همچنین متغیر محلی که در توابع استفاده می‌شود، حتما مقداردهی اولیه گردد.

۴–۵- عملگر مساوی یا مقدار دهی چندگانه (Multiple Assignments)

هنگامی که نیاز است از کلاس و یا استراکچر درون آرایه‌های (Array و List) استفاده گردد و یا نیاز است که بلادرنگ کپی از مقادیر یک کلاس و استراکچر را داشته باشیم، حتما باید از عملگر مساوی استفاده کنیم. همزمان با ایجاد عملگر مساوی، باید تابع مولد با ورودی مناسب نیز تعریف گردد.

از عمگرهای مساوی با ورودی‌های مختلف در مواقعی که نیاز به کپی متغیرهای مختلف باشد، استفاده می‌کنیم. برای مثال استراکچری را در نظر بگیرید که یک آرایه double دارد. عملگر مساوی اصلی با ورودی متغیری از نوع همان کلاس است. و عملگر مساوی با ورودی اشاره‌گر double می‌تواند برای کپی آرایه double بدرون کلاس بکار گرفته شود.

۵–۵- حلقه‌ها (for & while & do)

حلقه‌ها بمانند یک بلوک از کد می‌باشند. در صورت طولانی شدن و یا ناخوانائی بهتراست کامنت‌گذاری شوند. در اینجا به سه حلقه for و while و do می‌پردازیم. در حلقه‌ها عبارت چک‌شونده برای خروج از حلقه باید با شرایط مناسبی حتما رخ دهد. برای مثال در حلقه for معمولا index وظیفه شمردن تعداد loop حلقه را دارد و در صورت ردکردن از حد مجاز، از حلقه خارج می‌شود. در حلقه while هم بعداز تعداد loop لازم و برقراری شرط خروج از حلقه خارج می‌شود. توجه شود که حلقه do همانند while می‌باشد. با این تفاوت که شرط حلقه در انتهای حلقه چک می‌شود.

در حلقه‌ها از اشاره‌گر افزاینده یا کاهنده استفاده نگردد و بجای آن از روش index استفاده گردد. برای مثال بجای p++ از p[k] استفاده گردد. زیرا این خلاصه‌سازی هم خوانائی کد را کاهش می‌دهد و هم احتمال اشتباه و یا خطای کامپایلی بدلیل اشاره به حافظه نامعتبر را افزایش می‌دهد و هیچ بهبودی در کاهش زمان اجرا بوجود نمی‌آورد.

۶–۵- عبارت شرطی (if & switch)

برای نوشتن عبارت شرطی احتیاج به عبارت منطقی می‌باشد. در عبارت منطقی برای جلوگیری از خطای استفاده از یک مساوی بجای شرط‌مساوی (==) بهتر است متغیرها در سمت راست شرط‌مساوی قرار گیرند و تابع یا مقدار چک‌شونده در سمت چپ مساوی قرار گیرد. در عبارت منطقی از بکاربردن کپی متغیر بداخل متغیر دیگر و چک آن پرهیز شود.

در عبارت شرطی if سعی شود بخش شرط که توسط عبارت منطقی نگارش می‌شود، با پرانتزهای مناسب بخوبی از هم جدا شوند. اگر عبارت مثبت شرط طولانی بود، داخل پرانتز قرار گیرد. در شرطهای تودرتو حتما از پرانتزهای مناسب برای جدا کردن ifها استفاده گردد.

عبارت شرطی switch همانند چند if تودرتو می‌باشد. در عبارت شرطی switch هم باید بخش شرط با پرانتزهای مناسب بخوبی از هم جدا شوند. در caseهای داخل سوئیچ حتما از پرانتزهای مناسب و break برای جدا کردن caseها استفاده گردد.

۷–۵- متغیرهای پایه و typdef (bool, char, int, float, double, …)

در کدنویسی تاحد امکان از متغیرهای پایه استفاده گردد. و یا از typdefهای معروف و پرکاربرد استفاده گردد. در صورت تعریف typdef از تداخل و همنامی با typdefهای معروف و پرکاربرد پرهیز شود.

۸–۵- فاصله‌گذاری (Using White Space and Tab)

فاصله‌گذاری به خوانائی کد کمک می‌کند. قاعده کلی فاصله‌گذاری این است که با ایجاد هرپرانتز ({) حتما به اندازه یک Tab عبارت کد به سمت راست شیفت داده شود. توجه شود از بکاربردن فاصله space بجای فاصله Tab پرهیز شود.

از فاصله space برای جدا کردن عبارت‌های یک خط کد استفاده می‌شود. سعی شود توسط فاصله space محدوده عبارت منطقی و شروط آن‌را در عبارت منطقی تودرتو با پرانتزهای متعدد مشخص کرد. همچنین برای تعریف چند متغیر از یک نوع فاصله space مناسب قرار گیرد.