دانشکده‌ی مهندسی کامپیوتر

مبانی برنامه‌سازی کامپیوتر
تمرین شماره ۱ زبان C

سید صالح اعتمادی *

تاریخ اعلام: ۲۶ آذر ۱۳۹۸
مهلت تحویل: ۳ دی ۱۳۹۸

*تشکر ویژه از آقای علی حیدری که نسخه اولیه این قالب را در ترم دوم سال تحصیلی ۹۷-۹۸ برای درس برنامه‌سازی پیشرفته تهیه کردند.

فهرست مطالب

۱ آماده‌سازی

۱.۱ تعریف pipeline برای بیلد و تست برنامه‌های C

چنانچه قبلا pipeline برای بیلد و تست زبان C درست کرده‌اید به بخش بعدی مراجعه کنید. در غیر اینصورت ابتدا فایلی به نام `azure-pipelines-c.yml` در شاخه master با محتوای زیر درست کرده و add/commit/push کنید.

```
trigger:
- holymaster

pool:
  vmImage: 'ubuntu-latest'

steps:
- script: mkdir testout && for i in `find . -name '*_test.cpp'`; do g++
    -std=c++11 $i -o testout/$(basename $i .cpp); done
  displayName: 'build'

- script: for i in testout/*_test; do $i -d yes; done
  displayName: 'test'
```

برای تعریف pipeline به منظور بیلد و تست فایل‌های زبان C به Azure DevOps مراجعه کرده و پس از انتخاب گزینه `New Pipeline` در هنگام انتخاب نوع pipeline گزینه Existing Azure Pipelines YAML file را انتخاب کرده و فایل درست شده در بالا را از شاخه master انتخاب کنید. پس از درست شدن pipeline آن را به عنوان یک `Build Policy` برای شاخه holymaster تعیین کنید.

۲.۱ آماده‌سازی گیت

جدول ۱: قراردادهای نام‌گذاری

Naming conventions			
Feature Branch	Directory	Pull Request Title	Target Branch
fb_CA1	CA1	CA1	holymaster

✓ ابتدا به شاخه‌ی `master` بروید و از یکسان بودن این شاخه با سرور اطمینان حاصل کنید.

```
C:\git\FC98991>git checkout master
Already on 'master'
Your branch is up to date with 'origin/master'.

C:\git\FC98991>git status
On branch master
Your branch is up to date with 'origin/master'.

nothing to commit, working tree clean

C:\git\FC98991>git pull
Already up to date.

C:\git\FC98991>
```

✓ سپس این کار را برای شاخه `holymaster` تکرار کنید.

```
C:\git\FC98991>git checkout holymaster
Switched to branch 'holymaster'
Your branch is up to date with 'origin/holymaster'.

C:\git\FC98991>git status
On branch holymaster
Your branch is up to date with 'origin/holymaster'.

nothing to commit, working tree clean
```

```
C:\git\FC98991>git pull
Already up to date.

C:\git\FC98991>
```

✓ یک شاخه‌ی جدید با نام `fb_CA1` بسازید و تغییر شاخه دهید.

```
C:\git\FC98991>git branch fb_CA1

C:\git\FC98991>git checkout fb_CA1
Switched to branch 'fb_CA1'

C:\git\FC98991>git status
On branch fb_CA1
nothing to commit, working tree clean

C:\git\FC98991>
```

توصیه می‌شود پس از پیاده‌سازی هر تست تغییرات انجام شده را `commit` و `push` کنید. پوشه‌ای با نام `CA1` درست کرده و فایل‌های تست `bits_test.cpp` ، `memory_test.cpp` ، `string_test.cpp` به همراه فایل بستر تست `catch.hpp` را در آن قرار دهید. سپس پوشه `CA1` را با VSCode باز کنید.

۲ فعال‌سازی تست

سوال‌های تمرین بصورت تعدادی تست طراحی شده‌اند که لازم است تابع لازم برای پاس شدن تست را پیاده‌سازی کنید. همه تست‌ها comment شده و بستر Catch2 با استفاده از نشانه `[!hide]` برای رد کردن و عدم اجرای تست تنظیم شده است. ابتدا کامنت‌های مربوط به تست‌ها را یکی-یکی برداشته و از شناخته شدن تست توسط افزونه `Catch2 and Google Test Explorer` و `Test Explorer UI` در VSCode اطمینان حاصل کنید. برای این منظور ابتدا کامنت‌های مربوط به یک تست را بردارید. سپس تست را مطالعه کنید. نام تابع مورد تست و پارامترهای ورودی و نوع مقدار برگشتی تابع مورد تست را تشخیص دهید. سپس در فایل متناظر مربوطه `memory.h` ، `string.h` یا `bits.h` تابع را با پارمترهای ورودی و مقدار برگشتی مناسب تعریف کنید (در این مرحله نیاز به پیاده‌سازی نیست). در صورت نیاز می‌توانید فایلی به نام `memory.cpp` ، `string.cpp` یا `bits.cpp` نیز ایجاد کرده و تابع main را در آن پیاده‌سازی کرده و از درستی تعریف تابع خود مستقل از تست اطمینان حاصل نمایید. نهایتا به منظور شناخته شدن تست‌ها توسط VSCode لازم است فایل تست بیلد شود. همچنین لازم است تنظیم `catch2TestExplorer.executables` متعلق به `Catch2 and Google Test Explorer` به گونه‌ای تنظیم شده باشد که فایل‌های اجرایی حاصل از بیلد فایل‌های تست را پیدا کند: `bits_test.exe` ، `memory_test.exe` ، `string_test.exe` . این فرایند را تا براشته شدن کلیه کامنت‌های تست ادامه دهید.

۳ فرستادن تست در `Azure DevOps`

پس از شناخته شدن کلیه تست‌ها در VSCode کد خود را add/commit/push کرده و سپس در `Azure DevOps` یک `Pull Request` برای بردن این تغییرات از شاخه `fb_CA1` به شاخه `holymaster` درست کنید. چنانچه شاخه `holymaster` و `branch policy` مربوط به آنرا بدرستی تنظیم کرده باشید، بیلد مرتبط با این `Pull Request` باید موفقیت آمیز باشد.

۴ پیاده‌سازی توابع

از تست شماره یک شروع کرده ابتدا نشانه `[!hide]` را از تست حذف کرده تا تست فعال شود. سپس فایل تست را بیلد کرده و تست را اجرا کرده و از عدم اجرای موفقیت‌آمیز آن اطمینان حاصل کنید. سپس تابع مورد استفاده تست را بگونه‌ای پیاده‌سازی کنید که تست با موفقیت پاس شود.

۱.۴ تست‌های memory

تمرین‌های این بخش مربوط به اشاره‌گر، حافظه، متغیر و آرایه هستند. توابع مورد نیاز در این بخش را در فایل `memory.h` پیاده سازی کنید.

۱.۱.۴ تست variable swap

تابع `swap` اشاره‌گر به دو متغیر را به عنوان پارامتر دریافت کرده و مقدار دو متغیر را با هم جابجا می‌کند.

۲.۱.۴ تست byte of integer

تابع `get_byte` یک عدد صحیح مثبت و یک شماره بایت به عنوان پارامتر دریافت کرده و بایت مورد نظر را برمی‌گرداند.

۳.۱.۴ تست pointer math

تابع `address_plus` آدرس یکی از عناصر یک آرایه و عدد n به عنوان ورودی دریافت کرده و آدرس عنصری از آرایه که nتا بعد از آدرس ورودی قرار دارد را برمی‌گرداند.

۴.۱.۴ تست array as integer

تابع `array_as_int` یک آرایه از کاراکتر را بعنوان پارامتر دریافت کرده و چهار عنصر ابتدای آرایه را به عنوان یک عدد صحیح برمی‌گرداند.

۲.۴ تست‌های string

تمرین‌های این بخش مربوط به رشته یا آرایه کاراکتر می‌باشند. توابع مورد نیاز در این بخش را در فایل `string.h` پیاده سازی کنید.

۱.۲.۴ تست string length

تابع `str_len` یک آرایه از کاراکتر را بعنوان پارامتر دریافت کرده و طول رشته حرفی را برمی‌گرداند. کاراکتر صفر یا `NULL Terminator` جزو طول رشته حرفی حساب نمی‌شود.

۲.۲.۴ تست string compare

تابع `str_compare` دو آرایه از کاراکتر را بعنوان پارامتر دریافت کرده و در صورت برابر بودن دو رشته حرفی `true` و در غیر اینصورت `false` برمی‌گرداند.

۳.۲.۴ تست string append

تابع `str_append` دو آرایه از کاراکتر را بعنوان پارامتر دریافت کرده و دومین رشته حرفی را به انتهای اولین رشته حرفی اضافه می‌کند. توجه کنید که انتهای یک رشته حرفی با `NULL Terminator` یا کاراکتر صفر مشخص می‌شود.

۴.۲.۴ تست string copy

تابع `str_copy` دو آرایه از کاراکتر را بعنوان پارامتر دریافت کرده و اولین رشته حرفی را در آرایه دوم کپی می‌کند.

۳.۴ تست‌های bits

تمرین‌های این بخش مربوط به محاسبه یا تغییر بیت‌های یک عدد می‌باشد. توابع مورد نیاز در این بخش را در فایل `bits.h` پیاده سازی کنید.

۱.۳.۴ تست count ON bits

تابع `count_on_bits` یک عدد صحیح به عنوان پارامتر دریافت می‌کند و تعداد بیت‌های یک در نمایش مبنای دو این عدد را برمی‌گرداند.

۲.۳.۴ تست make bit stream

تابع `make_bits` اعداد صحیح n و m را به عنوان پارامتر برمی‌گرداند و یک عدد صحیح برمی‌گرداند که در نمایش مبنای دو n تا بیت یک داشته و بین هر دو بیت یک تعداد m تا بیت صفر موجود است. برای مثال به تست کیس‌ها مراجعه کنید.

۵ ارسال

اگر موفق به پاس شدن تستی نشدید دستور مربوط به عدم اجرای تست را قبل از تست باقی بگذارید. پس از پیاده‌سازی توابع و پاس شدن تست‌هایی که فرصت کردین، نوبت به ارسال آنها میرسد. مثل قبل تغییرات را در شاخه `fb_CA1` add/commit/push کنید.

۱.۵ ساخت Pull Request

با مراجعه به سایت Azure DevOps لز موفقیت بیلد برای Pull Request که در مرحله اول درست کردید اطمینان حاصل کنید و آنرا کامل کنید. دقت کنید که گزینه‌ی Delete source branch **نباید** انتخاب شود.