دانشکده‌ی مهندسی کامپیوتر

مبانی برنامه‌سازی کامپیوتر
آزمون عملی پایان ترم
پایتون و C

سید صالح اعتمادی *

۳۰ 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 نکات مورد توجه

- صدا و صفحه نمایش شما باید از طریق نرم‌افزار Flashback recorder به طور کامل از ابتدا تا انتهای امتحان ضبط و ذخیره شود. دقت کنید که پس از نصب نرم‌افزار، در قسمت تنظیمات کیفیت ضبط را ۱ فریم بر ثانیه قرار دهید. ویدیوی امتحان بعد از امتحان جمع‌آوری خواهد شد.
- استفاده از هرگونه منبع کاغذی، مجازی، کتابی، نوشتاری، ... در امتحان مجاز نمی‌باشد.
- دیدن هرگونه کد از روی اینترنت یا غیراینترنت مجاز نیست. پاسخ ارسالی هر کس حتما باید توسط خود او و بدون دیدن هیچ کد دیگری نوشته شده باشد. حتی اگر کد دیگر را خود فرد قبلا نوشته باشد. کمک گرفتن از دیگران در طول مدت امتحان مجاز نیست و منجر به درج نمره‌ی **مردود** برای این درس می‌شود.
- معیار ارزیابی امتحان فقط کدی است که در AzureDevOps با روشی که در ادامه آمده بارگزاری شده است.
- حین امتحان تنها اجازه ارتباط با استاد درس را دارید. هر گونه ارتباط با هر فرد دیگری در جلسه امتحان یا خارج از جلسه امتحان به صورت حضوری یا مجازی مجاز نمی‌باشد.
- در صورت نیاز به خروج از محل امتحان قبل از اتمام امتحان، امکان خروج بعد از هماهنگی با استاد و بدون بردن تلفن همراه و به صورت یک نفر، یک نفر هست.
- خوردن و آشامیدن در طول امتحان بدون برهم زدن نظم اشکال ندارد.

۲.۱ آماده‌سازی‌های اولیه

قواعد نام‌گذاری آزمون را از جدول ۱ مطالعه کنید.

جدول ۱: قراردادهای نام‌گذاری آزمون

Naming conventions			
Feature Branch	Directory	Pull Request Title	Target Branch
`fb_FinalExam`	`FinalExam`	`FinalExam`	`holymaster`

۱.۲.۱ آماده‌سازی‌های مربوط به git

اگر سر کلاس و کارگاه چند بار مفاهیم و روش کار با git آموزش داده شد اما بار دیگر در اینجا کارهایی را که باید در ابتدای آزمون انجام دهید را مرور می‌کنیم.

✓ 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_FinalExam` بسازید و تغییر شاخه دهید.

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

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

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

C:\git\FC98991>
```

توصیه می‌شود پس از پیاده‌سازی هر تست تغییرات انجام شده را `commit` و `push` کنید.

۲.۲.۱ آماده‌سازی‌های مربوط به VSCode

پوشه‌ای با نام `FinalExam` درست کرده و فایل‌های `exam_test.py`، `exam.py`، `exam.h`، `exam.cpp`، `main_test.cpp`، `catch.hpp` را در آن قرار دهید. همچنین به اختیار و مسولیت خودتان می‌توانید پوشه `.vscode` را نیز کپی کرده و از تنظیمات و task های از پیش تعریف شده آن استفاده کنید. سپس پوشه `FinalExam` را با VSCode باز کنید.

۲ پیاده‌سازی

۱.۲ آماده سازی تست

سوال‌های امتحان بصورت تعدادی تست به زبان پایتون در بستر pytest و به زبان C در بستر Catch۲ طراحی شده‌اند که لازم است تابع لازم برای پاس شدن تست را پیاده‌سازی کنید. همه تست‌ها comment شده و pytest و catch۲ برای رد کردن و عدم اجرای تست تنظیم شده. روش فعال‌سازی تست‌ها مطابق تمرین‌ها و امتحان‌های عملی قبلی می‌باشد. برای ارجاع، مستند امتحان عملی پایتون و تمرین عملی شماره ۲ زبان C ضمیمه شده‌اند. لازم است یک pipeline برای C و یک pipeline برای پایتون درست شده باشند و هر دو روی Build Policy شاخه `holymaster` فعال شده باشند. چنانچه یک بار این کار را قبلا انجام داده‌اید دیگر تکرار آن لازم نیست.

۱.۱.۲ 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_FinalExam` به شاخه `holymaster` درست کنید. چنانچه شاخه `holymaster` و `branch policy` مربوط به آنرا بدرستی تنظیم کرده باشید، هر دو بیلد مرتبط با این `Pull Request` باید موفقیت آمیز باشد.

۲.۲ پیاده‌سازی توابع

تمام سوال‌ها در زبان پایتون و زبان C باید پیاده‌سازی شده و تست‌های مربوطه پاس شوند. سعی شده سوال‌های راحت‌تر در ابتدا باشند. البته پیاده‌سازی برخی توابع در زبان C به کار بیشتری نیاز دارد. مطابق روش موجود در مستند تمرین‌ها و امتحان عملی قبل، تست‌ها را یکی-یکی از حالت کامنت خارج کرده و تابع لازم برای کامپایل موفقیت‌آمیز و شناخته شدن تست مربوطه را پیاده‌سازی کنید. سپس تست را فعال کرده و پیاده‌سازی تابع را کامل کنید.

۱.۲.۲ تست `q0_string_len`

تابع `string_len` یک رشته کاراکتری (در زبان C) یا string (در زبان پایتون) به عنوان پارامتر دریافت می‌کند و طول آن را برمی‌گرداند. استفاده از توابع از پیش تعریف شده برای پیدا کردن طول رشته در پایتون یا C مجاز نمی‌باشد.

```c
char pch1[7] = "123456";
REQUIRE(string_len(pch1) == 6);

char pch2[6] = "abcd0";
REQUIRE(string_len(pch2) == 5);

char pch3[1] = "";
REQUIRE(string_len(pch3) == 0);

char pch4[6] = "00000";
REQUIRE(string_len(pch4) == 5);
```

تست زبان C

```python
assert exam.string_len("123456") == 6
assert exam.string_len("abcd0") == 5
assert exam.string_len("") == 0
assert exam.string_len("00000") == 5
```

تست زبان پایتون

۲.۲.۲ تست `q1_str_cpy`

تابع `str_cpy` یک رشته حرفی پایان یافته با صفر (null terminated string) و یک buffer یا حافظه تخصیص یافته به عنوان پارامتر دریافت کرده و رشته حرفی را در حافظه تخصیص یافته کپی می‌کند. به خاطر تفاوت زبان‌ها تعریف تابع پایتون تفاوت جزئی دارد. به تست‌ها مراجعه کنید.

```c
char pch1[7] = "123456";
char pch1_copy[10];
str_cpy(pch1, pch1_copy);
REQUIRE(strcmp(pch1_copy, "123456") == 0);
pch1[0] = 'a';
REQUIRE(strcmp(pch1_copy, "123456") == 0);

char pch2[10] = "aabbccdw";
char pch2_copy[20];
str_cpy(pch2, pch2_copy);
REQUIRE(strcmp(pch2_copy, "aabbccdw") == 0);
pch2[4] = '1';
REQUIRE(strcmp(pch2_copy, "aabbccdw") == 0);
```

تست زبان C

```python
pch1 = "123456"
pch1_copy = exam.str_cpy(pch1)
assert pch1_copy == "123456"
pch1 = "a23456"
assert pch1_copy == "123456"

pch1 = "aabbccdw"
pch1_copy = exam.str_cpy(pch1)
assert pch1_copy == "aabbccdw"
pch1 = "aab1ccdw"
assert pch1_copy == "aabbccdw"
```

تست زبان پایتون

۳.۲.۲ تست `q2_char_count`

تابع `char_count` یک رشته حرفی و یک کاراکتر بعنوان ورودی دریافت کرده و تعداد تکرار حرف در رشته را برمی‌گرداند.

```
int count = char_count("ababa", 'a');
REQUIRE(count == 3);

count = char_count("ababa", 'b');
REQUIRE(count == 2);

count = char_count("123412340", '1');

REQUIRE(count == 2);

count = char_count("123412340", '0');
REQUIRE(count == 1);

count = char_count("123412340", 'a');
REQUIRE(count == 0);
```

تست زبان C

```
count = exam.char_count("ababa", 'a')
assert count == 3

count = exam.char_count("ababa", 'b')
assert count == 2

count = exam.char_count("123412340", '1')
assert count == 2

count = exam.char_count("123412340", '0')
assert count == 1

count = exam.char_count("123412340", 'a')
assert count == 0
```

تست زبان پایتون

۴.۲.۲ تست `q3_longest_char_sequence`

تابع `longest_char_sequence` یک رشته حرفی و یک کاراکتر بعنوان پارامتر دریافت کرده و طولانی‌ترین تعداد تکرار متوالی کاراکتر در رشته را برمی‌گرداند. بعنوان مثال برای رشته `aaaabaa` و کاراکتر `a` جواب درست ۴ می‌باشد.

```
int count = longest_char_sequence("abbbbaa", 'a');
REQUIRE(count == 2);

count = longest_char_sequence("abbbbaa", 'b');
REQUIRE(count == 4);

count = longest_char_sequence("aaaabaa", 'a');
REQUIRE(count == 4);

count = longest_char_sequence("aaaababaa", 'a');
REQUIRE(count == 4);

count = longest_char_sequence("aaaababaa", 'b');
REQUIRE(count == 1);

count = longest_char_sequence("abababababa", 'b');
REQUIRE(count == 1);
```

تست زبان C

```
count = exam.longest_char_sequence("abbbbaa", 'a')
assert count == 2

count = exam.longest_char_sequence("abbbbaa", 'b')
assert count == 4

count = exam.longest_char_sequence("aaaabaa", 'a')
assert count == 4

count = exam.longest_char_sequence("aaaababaa", 'a')
assert count == 4

count = exam.longest_char_sequence("aaaababaa", 'b')
assert count == 1

count = exam.longest_char_sequence("abababababa", 'b')
assert count == 1
```

تست زبان پایتون

۵.۲.۲ تست `q4_count_pattern`

تابع `count_pattern` یک رشته حرفی به عنوان متن و یک رشته حرفی به عنوان الگو در قالب پارامتر دریافت کرده و تعداد تکرار الگو در متن را برمی‌گرداند.

```
int count = count_pattern("aba", "ab");
REQUIRE(count == 1);

count = count_pattern("alibcalidefali", "ali");
REQUIRE(count == 3);

count = count_pattern("101101", "101");
REQUIRE(count == 2);

count = count_pattern("101010101", "101");
REQUIRE(count == 4);
```

تست زبان C

```
count = exam.count_pattern("aba", "ab")
assert count == 1

count = exam.count_pattern("alibcalidefali", "ali")
assert count == 3

count = exam.count_pattern("101101", "101")
assert count == 2

count = exam.count_pattern("101010101", "101")
assert count == 4
```

تست زبان پایتون

۶.۲.۲ تست `q5_function_pointer`

تابع `apply` دو لیست و یک تابع (با دو ورودی عدد صحیح و مقدار بازگشتی عدد صحیح) بعنوان پارامتر دریافت کرده و خروجی تابع بر روی هر زوج عنصر را در لیست سوم برمی‌گرداند. برای فهم بهتر تست‌ها را مطالعه کنید.

```
int list1[5] = {1, 3, 2, 5, 4};
int list2[5] = {1, 2, 0, 2, 0};
int sub_expected[5] = {0, 1, 2, 3, 4};
int add_expected[5] = {2, 5, 2, 7, 4};
```

```c
int sub_actual[5];
int add_actual[5];

apply(5, list1, list2, sub, sub_actual);
for(int i=0; i<5; i++)
    REQUIRE(sub_actual[i] == sub_expected[i]);

apply(5, list1, list2, add, add_actual);
for(int i=0; i<5; i++)
    REQUIRE(add_actual[i] == add_expected[i]);

int list21[3] = {0, 2, 5};
int list22[3] = {1, 3, 2};
int sub_expected2[3] = {-1, -1, 3};
int add_expected2[3] = {1, 5, 7};
int sub_actual2[3];
int add_actual2[3];

apply(3, list21, list22, sub, sub_actual2);
for(int i=0; i<3; i++)
    REQUIRE(sub_actual2[i] == sub_expected2[i]);

apply(3, list21, list22, add, add_actual2);
for(int i=0; i<3; i++)
    REQUIRE(add_actual2[i] == add_expected2[i]);
```

تست زبان C

```python
list1 = [1, 3, 2, 5, 4]
list2 = [1, 2, 0, 2, 0]
sub_expected = [0, 1, 2, 3, 4]
add_expected = [2, 5, 2, 7, 4]
sub_actual = []
add_actual = []

exam.apply(list1, list2, sub, sub_actual)
assert sub_actual == sub_expected

exam.apply(list1, list2, add, add_actual)
assert add_actual == add_expected

list1 = [0, 2, 5]
list2 = [1, 3, 2]
sub_expected = [-1, -1, 3]
add_expected = [1, 5, 7]
sub_actual = []
add_actual = []

exam.apply(list1, list2, sub, sub_actual)
assert sub_actual == sub_expected

exam.apply(list1, list2, add, add_actual)
assert add_actual == add_expected
```

تست زبان پایتون

۷.۲.۲ تست `q6_change_student`

ابتدا یک struct (در پایتون class) به نام `student` تعریف (typedef) کنید که فیلد اول آن یک رشته حرف به اندازه ۹ کاراکتر به نام `id` و فیلد دوم آن یک رشته حرفی به اندازه ۳۰ کاراکتر به نام `name` باشد. سپس دو تابع به نام‌های `change_student_id` و `change_student_name` تعریف کنید که اشاره‌گر به struct از نوع student و یک نام یا آی‌دی جدید به عنوان پارامتر دریافت کند و نام یا آی‌دی ساختار student را به مقدار جدید تغییر دهد. برای توضیح بیشتر تست را مطالعه کنید.

```c
student s = { "98521121", "Zhila Arghavan"};

char new_id[10] = "98521122";
change_student_id(&s, new_id);
REQUIRE(strcmp(new_id, s.id) == 0);

char new_name[30] = "Ali Mardani";
change_student_name(&s, new_name);
REQUIRE(strcmp(new_name, s.name) == 0);

char new_id2[10] = "96000001";
change_student_id(&s, new_id2);
REQUIRE(strcmp(new_id2, s.id) == 0);

char new_name2[30] = "Homa Sarbaz";
change_student_name(&s, new_name2);
REQUIRE(strcmp(new_name2, s.name) == 0);
```

تست زبان C

```python
s = exam.student( "98521121", "Zhila Arghavan")
new_id = "98521122"
exam.change_student_id(s, new_id)
assert new_id == s.id

new_name = "Ali Mardani";
exam.change_student_name(s, new_name)
assert new_name == s.name

new_id = "96000001"
exam.change_student_id(s, new_id)
assert new_id == s.id

new_name = "Homa Sarbaz";
exam.change_student_name(s, new_name)
assert new_name == s.name
```

تست زبان پایتون

۸.۲.۲ تست `q7_student_tostring`

تابع `q7_student_tostring` یک اشاره‌گر به ساختار student از ورودی دریافت کرده و یک رشته حرفی متناظر به صورت شماره‌دانشجویی، کاراکتر ':' و سپس نام را برمی‌گرداند. برای مثال تست‌ها را ببینید.

```c
student s = { "98521121", "Zhila Arghavan"};
char* str = student_tostring(&s);
REQUIRE(strcmp(str, "98521121:Zhila Arghavan") == 0);
free(str);

student s2 = { "98522321", "Keykhosro Ghobadi"};
char* str2 = student_tostring(&s2);
REQUIRE(strcmp(str2, "98522321:Keykhosro Ghobadi") == 0);
free(str2);
```

تست زبان C

```python
s = exam.student("98521121", "Zhila Arghavan")
str = exam.student_tostring(s)
assert str == "98521121:Zhila Arghavan"

s = exam.student("98522321", "Keykhosro Ghobadi")
str = exam.student_tostring(s)
assert str == "98522321:Keykhosro Ghobadi"
```

تست زبان پایتون

۹.۲.۲ تست `q8_create_student`

تابع `create_student` را بگونه‌ای تعریف کنید که یک شماره‌دانشجویی و نام بعنوان پارامتر دریافت کند و یک ساختار با نام و شماره‌دانشجویی داده‌شده برگرداند.

```c
char id1[9] = "98521121";
char name1[30] = "Zhila Arghavan";
student* ps = create_student(id1, name1);
REQUIRE(strcmp(id1, ps->id) == 0);
REQUIRE(strcmp(name1, ps->name) == 0);
free(ps);

char id2[9] = "98522321";
char name2[30] = "Keykhosro Ghobadi";
ps = create_student(id2, name2);
REQUIRE(strcmp(id2, ps->id) == 0);
REQUIRE(strcmp(name2, ps->name) == 0);
free(ps);
```

تست زبان C

```python
id = "98521121"
name = "Zhila Arghavan"
ps = exam.create_student(id, name)
assert id == ps.id
assert name == ps.name

id = "98522321"
name = "Keykhosro Ghobadi"
ps = exam.create_student(id, name)
assert id == ps.id
assert name == ps.name
```

تست زبان پایتون

۱۰.۲.۲ تست `q9_view_memory` : فقط برای زبان C

تابع `get_std_ptr1` و `get_std_ptr2` را بگونه‌ای تعریف کنید که تست پاس شود. مطالعه و فهم تست جزو سوال است. در صورت نیاز با استفاده از دستور `-exec x/32bx <address>` می‌توانید حافظه را مشاهده کنید.

```c
char id1[9] = "98521121";
char name1[30] = "Zhila Arghavan";
student* ps = create_student(id1, name1);
char* ptr1 = get_std_ptr1(ps);
char* ptr2 = get_std_ptr2(ps);
int w = *ptr1;
w <<= 8;
w |= *ptr2;
REQUIRE( w == 0x316c);
free(ps);

char id2[9] = "98522321";
char name2[30] = "Keykhosro Ghobadi";
ps = create_student(id2, name2);
ptr1 = get_std_ptr1(ps);
ptr2 = get_std_ptr2(ps);
w = *ptr1;
w <<= 8;
w |= *ptr2;
```

```
REQUIRE( w == 0x336b);
free(ps);
```

تست زبان C

۱۱.۲.۲ تست `q10_test_cos` : فقط برای زبان پایتون

تابعی به نام `cos` تعریف کنید که کسینوس یک عدد را تا دقت خواسته شده با استفاده از بسط تیلور حساب کند.

$$\cos(x) = ۱ - \frac{x^۲}{۲!} + \frac{x^۴}{۴!} - \frac{x^۶}{۶!} + \cdots$$

```
epsilon = 0.001
x = math.pi/2
cos_x = exam.cos(x, epsilon)
assert abs(cos_x-math.cos(x)) <= epsilon

x = math.pi/3
cos_x = exam.cos(x, epsilon)
assert abs(cos_x-math.cos(x)) <= epsilon
```

تست زبان پایتون

۱۲.۲.۲ تست `q11_test_arc_cos` : فقط برای زبان پایتون

تابعی بنویسید که با استفاده از تابع cos که در q10 تعریف کرده‌اید و روش جستجوی دو-دوئی (که در تمرین‌ها برای پیدا کردن جذر یک عدد توسط تابع توان انجام داده‌اید) ArcCos عدد ورودی را تا دقت خواسته شده حساب کند.

```
epsilon = 0.001
cos_x = 0.5
x = exam.arc_cos(cos_x, epsilon)
assert abs(x - math.acos(cos_x)) <= epsilon

cos_x = 0.3
x = exam.arc_cos(cos_x, epsilon)
assert abs(x - math.acos(cos_x)) <= epsilon
```

تست زبان پایتون

۳ 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_FinalExam` `add/commit/push` کنید.

۱.۳ 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.