



## مبانی برنامه‌سازی کامپیوتر تمرین زبان C

سید صالح اعتمادی \*

تاریخ اعلام: ۲۳ آذر ۱۴۰۱  
مهلت تحویل: ۳ دی ۱۴۰۱

---

\*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

۳	۱ آماده‌سازی
۳	۱.۱ تعریف pipeline برای بیلد و تست برنامه‌های C
۳	۲.۱ آماده‌سازی گیت
۳	۲ فعال‌سازی تست
۴	۳ فرستادن تست در Azure DevOps
۴	۴ پیاده‌سازی توابع
۴	۱.۴ تست‌های memory
۴	۱.۱.۴ تست variable swap
۴	۲.۱.۴ تست byte of integer
۴	۳.۱.۴ تست pointer math
۴	۴.۱.۴ تست array as integer
۴	۲.۴ تست‌های string
۴	۱.۲.۴ تست string length
۵	۲.۲.۴ تست string compare
۵	۳.۲.۴ تست string append
۵	۴.۲.۴ تست string copy
۵	۳.۴ تست‌های bits
۵	۱.۳.۴ تست count ON bits
۵	۲.۳.۴ تست make bit stream
۵	۴.۴ تست‌های myarray
۵	۱.۴.۴ تست sum array
۵	۲.۴.۴ تست sum array ptr
۵	۳.۴.۴ تست sum array 2d
۶	۴.۴.۴ تست sum array 2d ptr
۶	۵.۴.۴ تست sum 2d jagged array
۶	۶.۴.۴ تست sum 2d jagged array ptr
۶	۵.۴ تست‌های struct
۶	۱.۵.۴ تست simple struct
۶	۲.۵.۴ تست simple struct memory layout
۶	۳.۵.۴ تست complex struct memory layout
۶	۶.۴ تست‌های dynamic memory
۷	۱.۶.۴ تست repeat_value
۷	۲.۶.۴ تست range
۷	۷.۴ تست‌های function pointer
۷	۱.۷.۴ تست apply single one parameter function pointer
۷	۲.۷.۴ تست apply single two parameter function pointer
۷	۳.۷.۴ تست apply function pointer list to single value
۷	۴.۷.۴ تست return function pointer
۷	۵.۷.۴ تست return self struct with fn ptr
۷	۵ ارسال

## 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.

```

1 trigger:
2   - main
3
4 pool:
5   vmImage: 'ubuntu-latest'
6
7 steps:
8   - script: mkdir testout && for i in `find . -name '*_test.cpp`; do g++
9     -std=c++11 $i -o testout/$(basename $i .cpp); done
10     displayName: 'build'
11
12   - script: for i in testout/*_test; do $i -d yes; done
13     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 `main` تعیین کنید.

### ۲.۱ آماده‌سازی گیت

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

```

1 C:\git\FC01021>git checkout master
2 Already on 'master'
3 Your branch is up to date with 'origin/master'.
4
5 C:\git\FC01021>git status
6 On branch master
7 Your branch is up to date with 'origin/master'.
8
9 nothing to commit, working tree clean
10
11 C:\git\FC01021>git pull
12 Already up to date.
13
14 C:\git\FC01021>
```

✓

توصیه می‌شود پس از پیاده‌سازی هر تست تغییرات انجام شده را `commit` و `push` کنید. پوشه‌ای با نام `CA` درست کرده و فایل‌های تست `bits_test.cpp`، `memory_test.cpp`، `string_test.cpp`، `function_pointer_test.cpp`، `dynamic_memory_test.cpp`، `struct_test.cpp`، `myarray_test.cpp` به همراه فایل بستر تست `catch.hpp` را در آن قرار دهید. سپس پوشه `CA` 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 `doctest` با استفاده از نشانه `doctest::skip()` 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 `TestMate` و `Test Explorer` 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 `bits.h` ، `string.h` ، `memory.h` ، `function_pointer.h` یا `dynamic_memory.h` ، `struct.h` ، `myarray.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 `function_pointer.cpp` ، `dynamic_memory.cpp` ، `myarray.cpp` ، `bits.cpp` ، `string.cpp` ، `memory.cpp` نیز ایجاد کرده و تابع `main` را در آن پیاده‌سازی کرده و از درستی تعریف تابع خود مستقل از تست اطمینان حاصل نمایید. نهایتاً به منظور شناخته شدن تست‌ها توسط VSCode لازم است فایل تست بیلد شود. همچنین لازم است تنظیم `testMate.cpp.test.executables` متعلق به `TestMate` به گونه‌ای تنظیم شده باشد که فایل‌های اجرایی حاصل از بیلد فایل‌های تست را پیدا کند: `bits_test.exe` ، `dynamic_memory_test.exe` ، `myarray_test.exe` ، `string_test.exe` ، `string_test.exe` ، `memory_test.exe` ، `function_pointer_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` کنید. با توجه به اسکریپت موجود برای کامپایل و اجرای تست‌ها لازم است قبل از ارسال نهایی خط زیر را در ابتدای تمام فایل‌های `cpp` مربوط به `test` اضافه کنید تا فایل‌های تست به تنهایی کامپایل شده و فایل اجرایی متناظر با آنها در آژور تولید و اجرا شود.

```
#define DOCTEST_CONFIG_IMPLEMENT_WITH_MAIN
```

## ۴ پیاده‌سازی توابع

### 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` پیاده‌سازی 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  $n$  تا 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` پیاده‌سازی 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.

**۴.۴ تست‌های myarray**

تمرین‌های این بخش مربوط به پیمایش آرایه یک بعدی، دو بعدی و آرایه ناهمسان (`jagged array`) با عملگر `[]` و همچنین با استفاده از محاسبات اشاره‌گرها می‌باشد. توابع لازم برای این بخش را در فایل `myarray.h` پیاده سازی کنید.

در این بخش حتماً لازم است توسط دستور `-exec x/16bx <address>` در `DEBUG CONSOLE` موجود در `VS CODE` حافظه را در آدرس‌های مربوطه بررسی کنید.

**۱.۴.۴ تست sum array**

تابع `array_sum` یک آرایه و اندازه آن را به عنوان پارامتر دریافت کرده و جمع عناصر آنرا برمی‌گرداند.

**۲.۴.۴ تست sum array ptr**

تابع `array_sum_ptr` مانند تابع `array_sum` است، با این تفاوت که استفاده از عملگر `[]` در این تابع مجاز نمی‌باشد. لازم است با محاسبات اشاره‌گرها محل اعداد موجود در آرایه را پیدا کرده و حاصل جمع آنها را برگردانید. در صورت استفاده از عملگر `[]` از این سوال هیچ نمره‌ای دریافت نمی‌کنید.

**۳.۴.۴ تست sum array 2d**

تابع `array_sum2d` یک آرایه دوبعدی ۲ در  $n$  و  $n$  را به عنوان پارامتر دریافت کرده و حاصل جمع عناصر آرایه دوبعدی را برمی‌گرداند. در استاندارد C89 فقط بعد اول آرایه دو بعدی می‌تواند نامشخص باشد. بقیه ابعاد باید در زمان کامپایل مشخص باشند.

## ۴.۴.۴ تست sum array 2d ptr

تابع `array_sum2d_ptr` یک آرایه دوبعدی با ابعاد دلخواه را به صورت یک اشاره‌گر به عدد صحیح (`int *`) و اندازه بعد اول و دوم را به عنوان پارامتر دریافت کرده و حاصل جمع عناصر را برمی‌گرداند. همانند تابع `array_sum_ptr` استفاده از عملگر `[]` در این تابع مجاز نمی‌باشد.

## ۵.۴.۴ تست sum 2d jagged array

تابع `jagged_array_sum` یک آرایه دوبعدی ناهمسان به همراه اندازه بعد اول و آرایه‌ای یک بعدی شامل اندازه‌های مختلف بعد دوم را به عنوان ورودی دریافت کرده و حاصل جمع عناصر را برمی‌گرداند. دقت کنید که آرایه ناهمسان برخلاف آرایه دوبعدی، آرایه‌ای از آرایه‌ها (یا اشاره‌گرها) می‌باشد. حتما این تفاوت را با دستور مشاهده حافظه بررسی کنید.

## ۶.۴.۴ تست sum 2d jagged array ptr

تابع `jagged_array_sum_ptr` همانند تابع `jagged_array_sum` می‌باشد با این تفاوت که آرایه ناهمسان را به صورت (`int**`) و آرایه اندازه‌ها را به صورت (`int*`) به عنوان پارامتر دریافت می‌کند. همچنین استفاده از عملگر `[]` در این تابع مجاز نمی‌باشد. لازم است از محاسبات اشاره‌گرها استفاده شود.

## ۵.۴ تست‌های struct

تمرین‌های این بخش مربوط به تعریف `struct`، عملگر `.`، عملگر `->`، اشاره‌گر به `struct` و نحوه قرارگیری `struct` در حافظه می‌باشد. مانند بخش قبل:

**حتما لازم است توسط دستور `-exec x/16bx <address>` در `DEBUG CONSOLE` موجود در `VS CODE` به منظور بررسی حافظه استفاده کنید.**

## ۱.۵.۴ تست simple struct

ابتدا لازم است با استفاده از `typedef` یک نوع داده‌ای `struct` به نام `_student` با بخش‌های `name` و `grade` با نوع داده‌ای مناسب تعریف کنید. سپس تابعی به نام `add_grade` تعریف کرده که اشاره‌گر به `_student` و یک عدد `float` به عنوان پارامتر دریافت کند و مقدار عدد را به `grade` موجود در محل اشاره‌گر اضافه کند. همچنین لازم است تابع اشاره‌گر را نیز برگرداند.

## ۲.۵.۴ تست simple struct memory layout

ابتدا لازم است با استفاده از `typedef` یک نوع داده‌ای `struct` به نام `int_struct` با بخش‌ها و نوع‌های داده‌ای مناسب تعریف کنید. سپس تابعی به نام `get_some_ptr` تعریف کرده که اشاره‌گر به `int_struct` به عنوان ورودی دریافت کرده و یک اشاره‌گر `unsigned int*` به نحوی برمی‌گرداند که تست‌های آتی پاس شوند. برای پیدا کردن آدرس مناسب برای برگرداندن لازم است حافظه را با دستور بالا بررسی کنید.

## ۳.۵.۴ تست complex struct memory layout

در این تمرین نیز مانند تمرین قبل لازم است نوع داده‌ای مناسب را تعریف کرده و تابع `get_some_ptr2` را به گونه‌ای پیاده‌سازی کنید که تست‌ها پاس شوند. دقت کنید، همانند تمرین قبل، مطالعه تست، فهم آن و بررسی حافظه قسمت اصلی این تمرین می‌باشد. در این تمرین دقت کنید که هر کدام از بخش‌های `struct` در چه بخشی تعریف شده‌اند و فاصله خالی میان آنها را نیز پیدا کنید. برای فهم بهتر مفهوم `structure padding` به این آدرس مراجعه کنید.

## ۶.۴ تست‌های dynamic memory

تمرین‌های این بخش مربوط به تخصیص حافظه متغیر/پویا در `heap` می‌باشد. توابع مورد نیاز در این بخش را در فایل `dynamic_memory.h` پیاده‌سازی کنید. برای پیاده‌سازی توابع در این قسمت لازم است از تابع تخصیص حافظه پویا `malloc` استفاده کنید.

## ۱.۶.۴ تست repeat\_value

تابع repeat\_value یک عدد صحیح  $v$  و اندازه آرایه  $n$  به عنوان پارامتر دریافت کرده و یک آرایه با اندازه  $n$  که تمام عناصر آن دارای مقدار  $v$  می‌باشند، برمی‌گرداند.

## ۲.۶.۴ تست range

تابع range دو عدد صحیح from و to به عنوان پارامتر دریافت کرده و یک آرایه برمی‌گرداند که اعداد from تا to به ترتیب در آن قرار دارند.

## ۷.۴ تست‌های function pointer

تمرین‌های این بخش مربوط به اشاره‌گر به تابع می‌باشد. توابع مورد نیاز در این بخش را در فایل function\_pointer.h پیاده سازی کنید.

## ۱.۷.۴ تست apply single one parameter function pointer

تابع apply آدرس/اشاره‌گر به یک عدد صحیح به نام pn و یک اشاره‌گر به تابعی که ورودی و خروجی آن یکی عدد صحیح می‌باشد (pfn) از ورودی دریافت می‌کند. سپس تابع pfn را روی محتوای آدرس pn اجرا کرده و نتیجه را در همان محل pn ذخیره می‌کند.

## ۲.۷.۴ تست apply single two parameter function pointer

تابع apply2 دو عدد صحیح a و b و اشاره‌گر به یک عدد صحیح سوم pc به علاوه اشاره‌گر به یک تابع pfn به عنوان پارامتر دریافت می‌کند. سپس a و b را به تابع pfn داده و نتیجه را در محل pc ذخیره می‌کند.

## ۳.۷.۴ تست apply function pointer list to single value

تابع apply3 یک آرایه‌ای از توابع به همراه طول آریه و آدرس یک عدد صحیح را به عنوان پارامتر دریافت می‌کند. سپس توابع موجود در آرایه را به ترتیب یکی-یکی روی محتوای عدد صحیح اجرا کرده و نتیجه را در همان محل عدد صحیح ذخیره می‌کند.

## ۴.۷.۴ تست return function pointer

تابع get\_func یکی از کاراکترهای '+', '-', '/', '\*' را به عنوان پارامتر دریافت کرده و یک تابع (pfn) برمی‌گرداند. تابع pfn لازم است تابعی باشد که دو عدد از ورودی دریافت کرده و عملگر متناظر با کاراکتر ورودی را روی آنها اجرا کرده و نتیجه را برگرداند.

## ۵.۷.۴ تست return self struct with fn ptr

نوع داده‌ای struct با نام self را تعریف کنید که دو بخش داشته باشد. بخش اول یک عدد صحیح به نام n باشد. بخش دوم یک اشاره‌گر به تابع به نام f باشد. ورودی تابع f از نوع اشاره‌گر به self بوده و خروجی آن نیز از همین نوع می‌باشد. سپس تابع self\_func را تعریف کرده که تعریف آن مطابق تابع f باشد. پیاده‌سازی تابع self\_func باید به گونه‌ای باشد که تست‌های بعدی پاس شوند. با مطالعه تست‌ها پیاده‌سازی مناسب را ارائه کنید.

## ۵ ارسال

اگر موفق به پاس شدن تستی نشدید دستور مربوط به عدم اجرای تست را در تعریف TEST\_CASE مربوطه به شکل زیر اضافه کنید. پس از پیاده‌سازی توابع و پاس شدن تست‌هایی که فرصت کردین، نوبت به ارسال آنها میرسد. مثل قبل تغییرات را در شاخه master add/commit/push کنید.

```
TEST_CASE("test_case_name" * doctest::skip())
```