



FunctionCraft Programming Language Documentation

طراحی کامپایلر و زبان‌های برنامه‌نویسی

بهار ۱۴۰۳

دستیاران آموزشی: محمدجواد پسرکلو، علی دارابی، محمدعلی زمانی، محمد سوری،
علی عطااللهی، محمدرضا نعمتی

فهرست مطالب

2.....	۱) مقدمه
3.....	۱-۲) قواعد نحوی کلی
4.....	۲-۲) قواعد comment ها
4.....	۳-۲) قواعد نامگذاری متغیرها و function ها
5.....	۳) تعریف function ها
6.....	۴) فراخوانی function ها
7.....	۵) انواع داده
7.....	۶) پارامترها
8.....	۷) عملگرها
8.....	۱-۷) عملگرهای محاسباتی
9.....	۲-۷) عملگرهای مقایسه‌ای
9.....	۳-۷) عملگرهای منطقی
10.....	۴-۷) عملگر append
10.....	۵-۷) عملگرهای assignment
11.....	function pointer ها:
11.....	۶-۷) اولویت عملگرها
12.....	۸) گزاره‌های شرطی
13.....	۹) قواعد scoping
14.....	۱۰) function های پیش فرض
14.....	۱۱) حلقه‌ها (Loops)
16.....	۱۲) ساختار pattern matching

(۱) مقدمه

زبان برنامه‌نویسی **FunctionCraft** یک زبان functional است که برای انجام تمرین‌های عملی درس کامپایلر طراحی شده است. در این documentation، ساختار کلی برنامه‌های این زبان برنامه‌نویسی، قواعد نحوی آن، چگونگی تعریف function ها و پارامترهای آن‌ها توضیح داده می‌شوند. همچنین انواع داده (data types) و عملگرهای این زبان برنامه‌نویسی توضیح داده می‌شوند. در ادامه، ساختارهای control flow این زبان برنامه‌نویسی مانند حلقه‌ها و گزاره‌های شرطی معرفی می‌شوند. همچنین توابع پیش‌فرض، چگونگی تولید output و قواعد کار با string ها توضیح داده می‌شوند. برای فهمیدن راحت‌تر این مطالب، در قسمت‌های مختلف این documentation از مثال‌هایی استفاده شده است.

(۲) ساختار کلی

فایل‌های این زبان برنامه‌نویسی پسوند fl دارند. هر فایل از دو قسمت زیر تشکیل شده است:

- تعریف تعدادی function
- تعریف main function که entry point برنامه‌های نوشته شده به این زبان برنامه‌نویسی است.

در این زبان برنامه‌نویسی، متغیر global وجود ندارد. قطعه کد زیر، مثالی از ساختار کلی برنامه‌ها در این زبان برنامه‌نویسی است.



```
1 def get_size(i)
2     return len(g()[i]);
3 end
4
5 def g()
6     return [[1, 2]];
7 end
8
9 def main()
10     puts(get_size(0));
11 end
12
```

۱-۲) قواعد نحوی کلی

برخی از قواعد نحوی این زبان برنامه‌نویسی به صورت زیر می‌باشند:

- در انتهای هر گزاره (statement) یک semicolon وجود دارد.
- خط‌های خالی تاثیری در خروجی یا اجرای برنامه ندارند.
- همیشه main function در آخرین قسمت از برنامه نوشته می‌شود.
- در هر scope، پس از گزاره‌ی **return**، گزاره‌ی دیگری نمی‌تواند اضافه شود.
- ترتیب تعریف کردن function‌ها تاثیری در اجرای برنامه ندارد. در هر function می‌توان هر function دیگر را فراخوانی کرد، حتی اگر function فراخوانی شده در قسمتی از برنامه بعد از function فراخوانی‌کننده تعریف شده باشد.

۲-۲) قواعد comment ها

- در ابتدای comment های تک خطی کاراکتر # نوشته می شود. همه ی کاراکترهای بین کاراکتر # و انتهای خط، comment در نظر گرفته می شوند.
- در این زبان برنامه نویسی، comment های چندخطی با `=begin` شروع می شوند و با `=end` تمام می شوند. همه ی کاراکترهای بین `=begin` و `=end`، به عنوان comment در نظر گرفته می شوند.
- وجود comment ها در اجرای برنامه تاثیری ندارد. در ادامه، مثال هایی از comment های تک خطی و comment های چندخطی ارائه شده است:

```
1  =begin
2  The code defines a function called main
3  that prints "salam" to the console.
4  It contains comments marked with #,
5  which are explanations
6  ignored by the compiler.
7  =end
8
9  def main()
10     puts("salam"); # This is comment
11     # This is comment
12 end
```

۳-۲) قواعد نامگذاری متغیرها و function ها

نام متغیرها و function ها باید دارای ویژگی های زیر باشند:

- فقط شامل حروف کوچک (a...z)، حروف بزرگ (A...Z)، اعداد (0...9) و underscore باشند.

- نام متغیرها و function ها نمی‌توانند با عدد شروع شوند.
- keyword های زبان نمی‌توانند به عنوان نام متغیرها یا function ها به کار روند.
- همه‌ی keyword های زبان برنامه‌نویسی **FunctionCraft** در جدول زیر نوشته شده‌اند.

def	end	main	return
if	else	elseif	true
false	chop	chomp	push
puts	method	len	pattern
match	next	break	loop
do	for	in	

۳) تعریف function ها

- تعریف هر function با کلمه‌ی **def** شروع می‌شود. سپس نام function و پرانتزها نوشته می‌شوند. در صورتی که function دارای پارامتر باشد، پارامترها داخل پرانتز نوشته می‌شوند. در صورتی که بیش از یک پارامتر وجود داشته باشد، بین پارامترها comma نوشته می‌شود. تعریف function با کلمه‌ی **end** تمام می‌شود.
- main function پارامتر ندارد.
- در این زبان برنامه‌نویسی، نوعی از function ها به نام lambda functions وجود دارند و function هایی هستند که بدون نام تعریف می‌شوند. این function ها فقط می‌توانند در همان function ای که در آن تعریف شده‌اند، فراخوانی شوند. چگونگی تعریف lambda function به صورت زیر است:

-> (param1, param2, ...) { function_body }

در مثال زیر یک lambda function تعریف شده است که پارامتر a را می‌گیرد و a را return می‌کند:

-> (a) {return a;}

- Lambda function ها ممکن است پس از تعریف شدن فراخوانی شوند مانند مثال زیر:

```
1 -> (arg1, arg2) { return arg1 + arg2;} (1, 2);
```

در این مثال، lambda function تعریف شده و با آرگومان‌های 1 و 2 فراخوانی می‌شود.

- یک function ممکن است یک pointer به یک lambda function را return کند، مانند مثال زیر:

```
1 def foo()  
2     return -> (arg1, arg2) {return arg1 + arg2;};  
3 end
```

- یک function ممکن است بدون مقدار return کند.

۴) فراخوانی function ها

- فراخوانی همه‌ی function ها با استفاده از پرانتز انجام می‌شود. در صورتی که بیش از یک آرگومان ورودی داشته باشند، بین آرگومان‌ها comma نوشته می‌شود.
- برای استفاده از لیست‌ها به عنوان آرگومان، از reference آنها استفاده می‌شود. اما برای سایر انواع داده از مقدار آنها استفاده می‌شود.

۵) انواع داده

انواع primitive data type های موجود در این زبان در زیر مشخص شده اند:

- integer (*int*)
 - floating-point number (*float*)
 - string (*string*)
 - boolean value (*bool*)
 - list (*list*)
 - function pointer (*fptr*)
- مقادیر string حتما باید بین دو عدد double quotation نوشته شوند (مثال "String"). بنابراین مقادیر نوشته شده بین دو عدد single quotation به عنوان string پذیرفته نمی شوند.
 - همه‌ی element های لیست ها نوع داده‌ی یکسان دارند.
 - در این زبان برنامه نویسی، لیست های دوبعدی و لیست های چند بعدی هم وجود دارند. دسترسی به element های لیست با استفاده از index آنها انجام می شود.
 - برای ایجاد function pointer ها از کلمه‌ی method استفاده می شود و به صورت زیر است:

method(:function_name)

- فراخوانی function pointer ها مانند فراخوانی function های معمولی است.

۶) پارامترها

ممکن است برای پارامترهای function ها مقادیر پیش فرض تعیین شده باشد. در فراخوانی function ها، آوردن آرگومان هایی که مقدار پیش فرض دارند، اختیاری است. اگر این آرگومان ها در زمان فراخوانی function نوشته نشوند، از مقادیر پیش فرض آنها استفاده می شود. در مثال زیر، b و c مقادیر پیش فرض دارند.



```
1 def f2 (a, [b = 10, c = 20])
2     return a + b + c;
3 end
```

۷) عملگرها

در ادامه، عملگرهای این زبان برنامه‌نویسی توضیح داده می‌شوند:

۷-۱) عملگرهای محاسباتی

فقط اعداد می‌توانند به عنوان عملوند عملگرهای محاسباتی به کار روند. فهرست این عملگرها در جدول زیر ارائه شده است. مثال‌های این جدول با این فرض نوشته شده‌اند که $A=100$ و $B=10$ باشد.

عملگر	شرکت پذیری	توضیح	مثال
+	چپ	جمع	$A + B = 110$
-	چپ	تفریق	$A - B = 90$
*	چپ	ضرب	$A * B = 1000$
/	چپ	تقسیم	$A / B = 10$
-	راست	منفی تک عملوندی	$-A = -100$
--	چپ	پسوندی	$A--$
++	چپ	پسوندی	$A++$

۲-۷ عملگرهای مقایسه‌ای

این عملگرها دو مقدار را با یکدیگر مقایسه می‌کنند، بنابراین نتیجه‌ی آن‌ها از نوع boolean است. عملوندهای عملگرهای $<$ و $>$ و $<=$ و $>=$ باید از نوع int یا از نوع float باشند و نتیجه‌ی آن‌ها boolean است. فهرست عملگرهای مقایسه‌ای در جدول زیر ارائه شده است. در مثال‌های این جدول $A=100$ و $B=10$ می‌باشد.

عملگر	شرکت پذیری	توضیح	مثال
$<$ یا $<=$	چپ	کوچکتر	$A < B = \text{false}$
$>$ یا $>=$	چپ	بزرگتر	$A > B = \text{true}$

۳-۷ عملگرهای منطقی

عملوندهای عملگرهای منطقی از نوع boolean هستند. جدول زیر، فهرست عملگرهای منطقی را نشان می‌دهد. مثال‌های این جدول با این فرض هستند که $A=\text{true}$ و $B=\text{false}$ باشد.

عملگر	شرکت پذیری	توضیح	مثال
$\&\&$	چپ	عطف منطقی	$(A \&\& B) = \text{false}$
$\ \ $	چپ	فصل منطقی	$(A \ \ B) = \text{true}$
$!$	راست	نقیض منطقی	$(!A) = \text{false}$

۴-۷ عملگر append

از این عملگر برای اضافه کردن یک element به یک لیست استفاده می‌شود. نوع element اضافه شونده باید با نوع سایر element های موجود در آن لیست یکسان باشد. این عملگر، لیست را

به‌روزرسانی می‌کند و سپس آن را return می‌کند. ممکن است دو یا چند عملگر append به صورت متوالی به کار روند. این عملگر می‌تواند برای concatenate کردن string ها نیز به کار رود، مانند مثال زیر:



```
1 list << a << b << c;
2 puts("Hello" << " " << "World!");
```

۵-۷) عملگرهای assignment

عملوندهای دو طرف عملگرهای assignment باید دارای نوع یکسان باشند، در غیر این صورت type error ایجاد می‌شود. فهرست عملگرهای assignment در جدول زیر ارائه شده است.

عملگر	شرکت پذیری	توضیح	مثال
=	راست	مقدار عملوند سمت چپ را برابر مقدار عملوند سمت راست قرار می‌دهد.	A = B
+=	راست	مقدار عملوند سمت چپ را به مقدار عملوند سمت راست افزایش می‌دهد و مقدار جدید را جایگزین می‌کند.	A += B
-=	راست	مقدار عملوند سمت چپ را به مقدار عملوند سمت راست کاهش می‌دهد و مقدار جدید را جایگزین می‌کند.	A -= B
*=	راست	مقدار عملوند سمت چپ را در مقدار عملوند سمت راست ضرب می‌کند و مقدار جدید را جایگزین می‌کند.	A *= B
/=	راست	مقدار عملوند سمت چپ را در مقدار عملوند سمت راست ضرب می‌کند و مقدار جدید را جایگزین می‌کند.	A /= B

A %= B	مقدار باقی مانده عملوند سمت چپ را بر عملوند سمت راست حساب می‌کند و مقدار جدید را جایگزین مقدار عملوند سمت چپ می‌کند.	راست	%=
--------	--	------	----

function pointer ها :

با استفاده از عملگر assignment می‌توان یک function pointer را در یک متغیر ذخیره کرد. می‌توان با استفاده از آن متغیر و با اضافه کردن پرانتزها و آرگومانها پس از نام آن متغیر، function را فراخوانی کرد، مانند مثال زیر:

```

1 compare_ptr = method(:compare);
2 puts(compare_ptr(a, b));

```

اولویت عملگرها (۶-۷)

اولویت	دسته	عملگرها	شرکت پذیری
1	پرانتز	()	چپ به راست
2	دسترسی به عنصر های لیست	[]	چپ به راست
3	تک عملوندی پسوندی	++, --	چپ به راست
4	تک عملوندی پیشوندی	!, ~	راست به چپ
5	ضرب و تقسیم	*, /	چپ به راست
6	جمع و تفریق	+, -	چپ به راست

چپ به راست	<=,>=,<, >	مقایسه	7
چپ به راست	==, !=	مقایسه تساوی	8
چپ به راست	&&	عطف منطقی	9
چپ به راست		فصل منطقی	10
راست به چپ	=	تخصیص	11
چپ به راست	<<	الحاق	12

(۸) گزاره‌های شرطی

- شرط گزاره‌های if و حلقه‌ها الزاما از نوع boolean نیست.
- شرط if و حلقه‌ها باید دارای پرانتز باشد. ممکن است بین پرانتزها از عملگرهای منطقی (مانند && و ||) استفاده شده باشد. بنابراین شرط if و while حلقه‌ها از یک یا تعداد بیشتری مجموعه پرانتز و عملگرهای منطقی بین آن‌ها تشکیل می‌شود.

```

1  if (a == 3) && (b == 1)
2      puts(a + b);
3      if (a == 3) && ((b == 1) || (c == 5))
4          puts(a + b + c);
5      end
6  end

```

گزاره‌ی زیر در زبان C مفروض است:

if(e1 && (e2 || e3))

این گزاره در زبان **FunctionCraft** به صورت زیر نوشته می شود:

`if (e1) && ((e2) || (e3))`

- ممکن است block های **if** و **elseif** و **else** به صورت متوالی به کار روند. در این صورت در پایان این ساختار شرطی، حتما کلمه **end** نوشته می شود.

```
1  if (a)
2    ...
3  elseif (b)
4    ...
5  elseif (c)
6    ...
7  else
8    ...
9  end
```

- ممکن است ترکیب های مختلفی از **if** با **elseif** و **else** به کار رود. اما به ازای هر **if**، حداکثر یک block از نوع **else** می تواند وجود داشته باشد.
- در انتهای هر یک از block های **if** و **elseif** و **else** حتما باید کلمه **end** نوشته شده باشد.

۹) قواعد scoping

در این زبان برنامه نویسی، scope های جدید به صورت زیر ایجاد می شوند:

- آرگومان ها و خطوط کد داخل یک متد
- خطوط کد داخل گزاره های decision making مانند if و حلقه ها

در این زبان، برخی از قواعد scoping به صورت زیر می باشند:

- متغیرهای تعریف شده داخل یک function، فقط در scope های داخلی تر قابل دسترسی هستند، اما در بیرون از آن function قابل دسترسی نیستند.
- نمی توان در یک scope، آرگومان های همانم تعریف کرد.
- ممکن است گزاره ی return به صورت اختیاری در اطراف مقدار بازگشتی پرانتز داشته باشد.


۱۰ function های پیش فرض

در این زبان برنامه نویسی، تعدادی function به صورت پیش فرض تعریف شده اند که در ادامه توضیح داده می شوند:

- puts: متغیرها و متن را در خروجی چاپ می کند. مقادیر boolean به صورت 1 و 0 چاپ می شوند.
- push: یک element را به یک لیست یا یک کاراکتر را به یک string اضافه می کند.
- Len: طول یک لیست (تعداد element های آن) یا طول یک string (تعداد کاراکترهای آن) را return می کند.
- main: به عنوان entry point همه ی برنامه های نوشته شده به زبان برنامه نویسی FunctionCraft می باشد.
- chop: آخرین کاراکتر موجود در یک string را حذف می کند و string باقیمانده را return می کند.
- chomp: همه ی کاراکترهای newline (یعنی \n) موجود در یک string را حذف می کند و string باقیمانده را return می کند.

۱۱ حلقه ها (Loops)

- در این زبان برنامه نویسی، ساختار **do loop** تعریف شده که شرط ندارد و scope آن با کلمه ی end تمام می شود.



```

1  loop do
2      puts("start");
3      ...
4      puts("end");
5  end

```

- در این زبان برنامه‌نویسی، ساختار **for ... in ...** تعریف شده است که scope آن با کلمه‌ی end تمام می‌شود. این ساختار برای list ها یا range کاربرد دارد. یک range به صورت زیر تعریف می‌شود:

(start..end)


- در تعریف این بازه، start و end اعداد صحیح (**int**) هستند. این بازه از چپ بسته و از راست باز است. (low (inclusive) to high (exclusive))
- ساختارهای **break** و **break if** و **next** و **next if** می‌توانند داخل حلقه‌های **loop do** و **for** به کار روند.
- با اجرا شدن ساختار **break**، برنامه از حلقه کنونی خارج می‌شود.
- با استفاده از ساختار **break if** در صورتی که شرط ذکر شده برقرار باشد، از حلقه خارج می‌شود.
- با اجرا شدن ساختار **next**، برنامه از همان نقطه به iteration بعدی از این حلقه می‌رود.
- با استفاده از ساختار **next if** در صورتی که شرط ذکر شده برقرار باشد، از همان نقطه به iteration بعدی از این حلقه می‌رود.



```
1 list = [1, 2, 3, 4, 5, 6, 7, 8, 9];
2 for num in list
3     next if (num / 2 == 1);
4     puts(num);
5 end
6
7 start = 0;
8 finish = 9;
9 for num in (start..finish)
10     next if (num / 2 == 1);
11     puts(num);
12 end
```

۱۲ ساختار pattern matching

- این ساختار به شبیه ساختار switch case می باشد. ساختار pattern matching یک یا چند ورودی را می گیرد و شبیه یک function، بر اساس مقادیر ورودی یک خروجی return می کند.
- پس از هر کاراکتر '|', یک گزاره ی شرطی شبیه شرط if نوشته می شود. سپس، بعد از کاراکتر '=', یک مقدار نوشته می شود. بنابراین در صورت صحیح بودن شرط، مقدار نوشته شده جلوی آن return می شود.
- قبل از هر condition، باید indentation (به مقدار یک کاراکتر tab یا چهار کاراکتر space) وجود داشته باشد. مثال:



```

1  pattern fib(n)
2      | (n == 0) = 1
3      | (n == 1) = 1
4      | (n > 2) = fib (n-1) + fib (n-2)
5  ;
6
7  def main()
8      fib_5 = fib.match(5);
9  end
10

```

- برای فراخوانی یک pattern بر روی input ها از syntax زیر استفاده می شود:

pattern_name.match(arg1, arg2,...)

مثال:



```

1  fib_5 = fib.match(5);

```