سند زبان برنامه‌نویسی

C--

فهرست

1. مقدمه

زبان ابداعی --C یک زبان دستوری[1] است و در آن قابلیت تعریف ساختمان[2] وجود دارد. در این زبان یک بخش اصلی به نام main وجود دارد که اجرای برنامه از آن آغاز می‌شود. برنامه‌هایی که در این زبان نوشته می‌شوند، در هنگام اجرا دستورات درون این بخش را اجرا می‌کنند.

2. ساختار کلی

در این زبان، کد برنامه درون یک فایل با پسوند cmm قرار دارد. این فایل از قسمت‌های زیر تشکیل شده است:

- تعریف ساختمان‌ها
- تعریف توابع
- یک بخش main

یک نمونه از کد در این زبان به صورت زیر است:

```
struct Student begin
    int grade
    bool pass
end

bool check_pass (struct Student std)
    return std.pass

main() begin
    struct Student std;
    display (check_pass(std));
end
```

[1] Imperative
[2] Structure

2-1. قواعد کلی نحو

زبان --C به بزرگ و کوچک بودن حروف حساس است. در این زبان، وجود کاراکترهای Tab و Space تاثیری در خروجی برنامه ندارد. جزئیات مربوط به Scopeها و خطوط برنامه در ادامه به طور مفصل توضیح داده خواهد شد.

2-2. کامنت‌ها

در این زبان، امکان تعریف کامنت‌ها به صورت چند خطی وجود دارد. یک کامنت با */ شروع و با /* پایان می‌پذیرد. تمامی کاراکترهای بین این دو هیچ تاثیری در خروجی برنامه ندارند.

```
main() begin
    /* Welcome to Compiler course
       This is just a comment!
    */
    display (1400)
end
```

2-3. قواعد نام‌گذاری ساختمان‌ها، توابع و متغیرها

اسامی انتخابی برای نام‌گذاری ساختمان‌ها، توابع و متغیرها باید از قواعد زیر پیروی کنند:

- تنها از کاراکترهای a..z، A..Z، _ و ارقام تشکیل شده باشند.
- با رقم شروع نشوند.
- معادل کلیدواژه‌ها نباشند. در جدول زیر تمام کلیدواژه‌های این زبان آمده است:

struct	main	int	bool	list
void	while	do	if	else
return	get	set	begin	end
display	append	size	true	false
fptr				

- نام هر تابع یکتاست.
- نام هر ساختمان یکتاست.
- نام هر متغیر در یک Scope یکتاست.
- در Scope درونی امکان تعریف متغیر همنام با متغیری در Scope بیرونی وجود ندارد.

3. تعریف بخش main و توابع

main به صورت زیر تعریف می شود:

```
main() begin
   /*body*/
end
```

main مقدار بازگشتی ندارد و بدنه آن هم مانند یک تابع تعریف می‌شود که در ادامه توضیح داده خواهد شد.

تعریف یک تابع به صورت زیر انجام می‌شود:

```
/*
“return type”  “name of function” (arguments) begin
   body
end
 */

int sampleFunction (int arg1, int arg2)  begin
     /*body*/
end
```

همه توابع باید مقدار بازگشتی داشته باشند و در صورتی که مقدار بازگشتی توابع از نوع void نباشد، باید یک دستور return در تابع وجود داشته باشد.

فراخوانی توابع، تخصیص، اضافه کردن عضو به لیست، تعریف متغیر و return گزاره[3]های این زبان محسوب می‌شوند. هر گزاره در یک خط جداگانه نوشته می‌شود و گذاشتن کاراکتر ; در انتهای خط اختیاری است. هم چنین امکان نوشتن چند گزاره در یک خط وجود دارد که در این صورت در انتهای هر گزاره باید کاراکتر ; قرار بگیرد.

یک بلاک با کلیدواژه begin شروع شده و انتهای آن با کلیدواژه end مشخص می شود. در صورتی که در یک Scope چند گزاره داشته باشیم، باید درون بلاک قرار بگیرند؛ در غیر این صورت تعریف بلاک اختیاری است. یک نمونه مثال در زیر آمده است :

```
int sampleFunction (int arg1, int arg2)
    return arg1

main ()
    display (sampleFunction(1, 2))
```

4. ساختمان

ساختمان یک نوع داده‌ای است که در آن می توان متغیرهایی از انواع مختلف (به جز ساختمان از همان نوع) ذخیره کرد. ساختمان‌ها در ابتدای برنامه تعریف می‌شوند و با کاراکتر . می‌توان به عنصر آن دسترسی پیدا کرد. در مثال زیر نحوه تعریف یک ساختمان نشان داده است:

```
struct Point begin
      int x;
      int y;
      struct Neighbor neighbor;
end
```

[3] Statement

در تعریف ساختمان می توان به متغیرهای آن مقدار اولیه داد و پس از آن در طی اجرای برنامه مقدار این متغیر امکان عوض شدن ندارد.

برای هر متغیر (به جز آن‌هایی که مقدار اولیه دارند) می توان دو تابع getter و setter تعریف کرد که حتما در یک بلاک دارای begin و end قرار دارند(خط ۴ و ۱۱ در مثال زیر) . همچنین دو تابع getter و setter به تنهایی نمی‌آیند و setter بالاتر از getter تعریف می‌شود.

درون این توابع فقط به متغیرهای تعریف شده در ساختمان دسترسی وجود دارد. توجه شود که در هیچ یک از این دو تابع امکان تعریف متغیر جدید وجود ندارد. به مثال زیر توجه کنید:

```
struct Rectangle begin
    int length
    int width
    int middle (int newL, int newW) begin
        set begin
            length = length + newL / 2;
            width = width + newW / 2;
        end
        get
            return length / 2 + width / 2
    end
end
main() begin
    struct Rectangle rec
    rec.length = 10;rec.width = 2
    display(rec.length); display(rec.width);
    rec.middle = (4, 2)
    display(rec.middle)
    display(rec.length);display(rec.width)
end
```

پس از اجرای قطعه کد بالا، خروجی به صورت زیر نمایش داده می‌شود:

```
10
2
7
12
3
```

5. انواع داده

در زبان --C، دو نوع پایه int و bool و دو نوع دیگر list و fptr ،که توضیحات آن‌ها در ادامه آمده است، وجود دارند. در متغیرهای از نوع تایپ‌های پایه، خود مقادیر ذخیره می شود و نه اشاره‌گر به خانه‌ای از حافظه. هم‌چنین تایپ void نشانگر مقدار بازگشتی توابع بدون خروجی است. استفاده از این تایپ به عنوان عملوند و تعریف متغیر از این تایپ غیرمجاز است.

5-1. لیست

لیست دارای تعدادی عنصر است. در این زبان، اعضای لیست باید حتما از یک نوع باشند. نوع عناصر لیست می‌تواند یکی از چهار نوع int، bool، struct یا list باشد. در هنگام تعریف لیست، یک لیست خالی ایجاد می‌شود و نمی‌توان به آن مقدار اولیه داد. عناصر، توسط تابع append به لیست اضافه می شوند.

مثال :

✓	[1, 2, 3]
✓	[[[1, 2], [3, 4], [5,6]], [[1], [2, 3]]]
✗ **عضو اول لیستی از لیست‌هاست اما عضو دوم لیستی از int می‌باشد . پس نوع اعضا یکسان نیست.**	[[[1, 2], [3, 4], [5,6]], [1, 2, 3]]

دو تابع پیش‌فرض size و append مخصوص لیست‌ها می‌باشند که توضیحات مربوطه در توابع پیش‌فرض آمده است.

2-5. اشاره گر به توابع (fptr)

Fptr ها متغیر هایی از جنس پوینتر می‌باشند که به توابع برنامه اشاره می‌کنند و می‌توان توابع را به صورت غیرمستقیم توسط آن‌ها فراخواند. به طور مثال یک متغیر از جنس <int -> bool> به تابع با ورودی از جنس int و خروجی bool اشاره می‌کند. برای توصیف تابعی بدون ورودی و خروجی از <void -> void> استفاده می‌کنیم. نحوه تعریف و استفاده از آن‌ها در زیر آمده است :

```
int f1 (int arg1, int arg2)  begin
   return (arg1 * arg2) - (arg1 + arg2)
end
main() begin
      fptr  < int, int -> int > fpt1  = f1
      fptr  < int, int -> int > fpt2
      fpt2 = fpt1
      display(fpt1(5, 10))
      display(fpt2(15, 19))
end
```

6. متغیرها

تعریف متغیرها به صورت زیر انجام می گیرد:

```
/* bool or int:  type + name
    list : list + # + type + name
*/
int a
bool d
list # int b
list # list # int c
```

در این زبان چندین متغیر را می توان در یک گزاره تعریف کرد و به آن ها مقدار اولیه داد.

```
int a, b, c
int num = 10, i
```

در صورتی که به متغیری مقداری نسبت داده نشود، مقدار آن برابر با مقدار پیش فرض نوع خود در نظر گرفته می شود. مقادیر پیش فرض نوع های مختلف در جدول زیر مشخص شده است.

bool	false
int	0
list	[]
fptr	null

7. عملگرها

عملگرها در این زبان به چهار دسته‌ی عملگرهای حسابی، مقایسه‌ای، منطقی و عملگر تخصیص تقسیم می‌شوند.

7-1. عملگرهای حسابی

این دسته از عملگرها تنها روی اعداد عمل می‌کنند، لیست این عملگرها در جدول زیر آمده است.

در مثال‌های استفاده شده A برابر 20 و B را برابر 10 در نظر بگیرید:

عملگر	شرکت‌پذیری	توضیح	مثال
+	چپ	جمع	A + B = 30
-	چپ	تفریق	A – B = 10
*	چپ	ضرب	A * B = 200
/	چپ	تقسیم	A / B = 2 B / A = 0
-	راست	منفی تک‌عمل‌وندی	A = -20

7-2. عملگرهای مقایسه‌ای

این عملگرها وظیفه‌ی مقایسه را دارند، پس نتیجه‌ی آن‌ها باید مقدار صحیح یا غلط (true, false) **باشد. با این حساب خروجی این عملگرها یک** bool **است.**

توجه داشته باشید که عملوند عملگرهای > **و** < **تنها از جنس عدد صحیح هستند. همچنین برای عملگر** == **نیز باید نوع عملوندها یکسان باشند؛ در غیر اینصورت باید خطای کامپایل گرفته شود.**

لیست عملگرهای مقایسه‌ای در جدول زیر آمده است. در مثال‌های استفاده شده مقدار A را برابر 20 و مقدار B را برابر 10 بگیرید:

عملگر	شرکت‌پذیری	توضیح	مثال

==	چپ	تساوی	(A == B) = false
>	چپ	کوچکتر	(A < B) = false
<	چپ	بزرگتر	(A > 20) = true

3-7. عملگرهای منطقی

در زبان C-- عملیات منطقی تنها روی نوع داده‌ی bool قابل اعمال است.این عملگرها در جدول زیر لیست شده‌اند. در مثال‌های استفاده شده A را برابر true و B را برابر false در نظر بگیرید:

عملگر	شرکت‌پذیری	توضیح	مثال
&	چپ	عطف منطقی	(A & B) = false
|	چپ	فصل منطقی	(A | B) = true
~	راست	نقیض منطقی	~A = false

4-7. عملگر تخصیص

این عملگر که به صورت = نمایش داده می‌شود، وظیفه‌ی تخصیص را برعهده دارد. یعنی مقدار عملوند سمت راست را به عملوند سمت چپ اختصاص می‌دهد.

دقت داشته باشید که عملوند سمت چپ باید از نوع Lvalue باشد. عبارات Lvalue عباراتی هستند که به یک مکان در حافظه اشاره میکنند. در مقابل عبارات Rvalue به مکان خاصی در حافظه اشاره نمی‌کنند و صرفا یک عبارت دارای مقدار هستند. به عنوان مثال یک متغیر یا یک دسترسی به یکی

از عناصر لیست یک عبارت Lvalue است اما عبارت 1+5 یک عبارت Rvalue محسوب می‌شود.
عبارات Rvalue تنها در سمت راست عملگر تخصیص قرار می‌گیرند.

5-7. اولویت عملگرها

اولویت عملگرها طبق جدول زیر است:

اولویت	دسته	عملگرها	شرکت‌پذیری
1	پرانتز	( )	چپ به راست
2	دسترسی به اعضاء	.	چپ به راست
3	دسترسی به عناصر لیست	[]	چپ به راست
5	تک عملوندی	~ -	راست به چپ
6	ضرب و تقسیم	* /	چپ به راست
7	جمع و تفریق	+ -	چپ به راست
8	رابطه‌ای	< >	چپ به راست
9	تساوی	==	چپ به راست
10	عطف منطقی	&	چپ به راست
11	فصل منطقی	|	چپ به راست
12	تخصیص	=	چپ به راست
13	کاما	,	چپ به راست

8. ساختار تصمیم‌گیری

در زبان C-- تنها ساختار تصمیم‌گیری، if... else است. همچنین ساختار if می‌تواند بدون else استفاده گردد:

```
if (a == 2) begin
  /* some statements */
end
```

```
if (a == 2) begin
  /* some statements */
end
else begin
  /* some statements */
end
```

9. ساختار تکرار

در این زبان از while و do...while به عنوان ساختار تکرار استفاده می شود. گزاره های درون بلاک do تنها یک بار اجرا می شوند و پس از آن درستی عبارت پس از while سنجیده می شود.

```
while (a > 0) begin
      /* some statements */
end
```

```
do begin
    /* some statements */
end while (a > 0)
```

10. Scopeها

10-1. Scope های موجود در زبان

به طور کلی در زبان C-- موارد زیر در Scope جدیدی قرار دارند:

1. **آرگومان‌ها و خطوط کد داخل یک تابع**
2. **خطوط کد داخل گزاره های تصمیم گیری و حلقه تکرار**
3. **خطوط داخل یک ساختمان**

10-2. قوانین scopeها

نکات زیر در مورد Scopeها وجود دارد:

- **متغیرهایی که داخل یک تابع تعریف می‌شوند در Scopeهای بیرون آن دسترس‌پذیر نیستند و صرفاً در Scopeهای درون آن قابل دسترسی هستند.**
- **امکان تعریف متغیر با نام یکسان در یک Scope و Scope های درونی آن وجود ندارد.**
- **خطوط خالی از کد اجرایی هیچ تاثیری در خروجی و اجرای برنامه ندارد.**

11. توابع پیش‌فرض

11-1. تابع display

این تابع یک مقدار int یا bool دریافت می‌کند و آن را در خروجی چاپ می‌کند. مثال :

```
display (1)
```

11-2. تابع size

این تابع یک لیست را به عنوان آرگومان می‌گیرد و تعداد اعضای آن لیست را نمایش می‌دهد.

مثال :

A = [1, 2, 3, 4]

size(A) = 4

11-3. تابع append

این تابع یک لیست را به عنوان آرگومان اول و سپس عضوی که قرار است به لیست اضافه شود را به عنوان آرگومان دوم میگیرد و آن را اضافه میکند . در نهایت لیست تغییر کرده و عضو جدید را نیز شامل می‌شود. تایپ بازگشتی این تابع void می‌باشد.

مثال :

List before append	append	List after append
A = [1, 2, 3, 4]	append(A, 5)	A = [1, 2, 3, 4, 5]
B = [1, 2, 3, 4]	append(B, 3+4)	B = [1, 2, 3, 4 , 7]
C = [[1,2], [3,4]]	append(C, [5,6])	C = [[1, 2], [3, 4], [5, 6]]
D = [[1,2], [3,4]]	append(D[0], 5)	D = [[1, 2, 5], [3, 4]]

12. یک نمونه کد در این زبان

```
struct Person begin
  int age;
  int weight;
  int id;
end

void print_id(list# struct Person person) begin
   int i
   i = 0
   do begin
      display (person[i].id)
      i = i + 1
   end
   while i < n
end

main() begin
  int i, n = 10;
  list #struct Person people;

  while ~ (i == n) begin
      struct Person new_person
      new_person.id = i
      append(people, new_person)
      i = i + 1
   end
  fptr <list #struct Person -> void> pointer = print_id
  pointer(people)
end
```