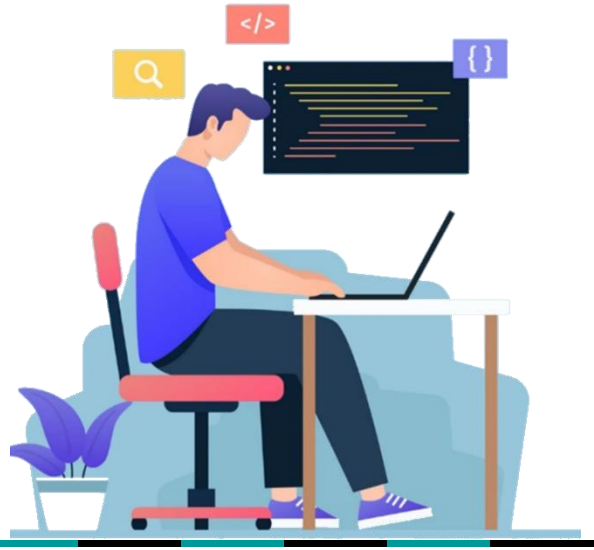
اصول و مبانی برنامه‌نویسی



مجید شبیری

کارشناسی ارشد IT، گرایش شبکه

از دانشگاه صنعتی امیرکبیر

ساختمان داده ها و ساختارهای کدنویسی

سرفصل مطالب

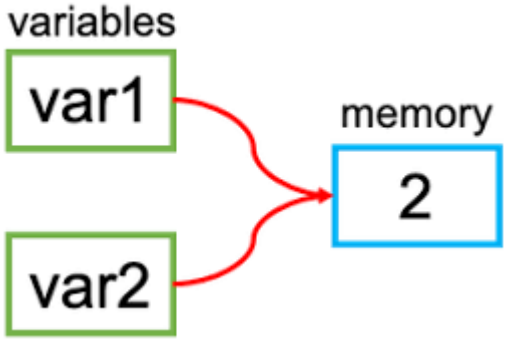
ساختمان داده‌ها و ساختارهای کدنویسی

زبان‌های برنامه‌نویسی نیز مانند زبان‌های محاوره ای (مانند فعل، اسم، صفت، پیشوند) دارای اجزا و ساختارهای مشخصی هستند :

- **متغیرها (Variables)**
  - دیتاتایپ (Data Types)
  - اعداد (Number)
  - کاراکتر (char)
  - آرایه (Array)
  - رشته (String)
- **عملگرها (operators)**
  - محاسباتی (+ , - , * , / , %)
  - منطقی (! , || , &&)
  - مقایسه ای (=! , == , >= , > , <=, <)
- **ساختارهای کنترل (تصمیم گیری)**
  - if
  - If-else
  - If-elseif
  - switch
- **ساختارهای تکرار (حلقه)**
  - while
  - do-while
  - for
- **فانکشن (Function)**

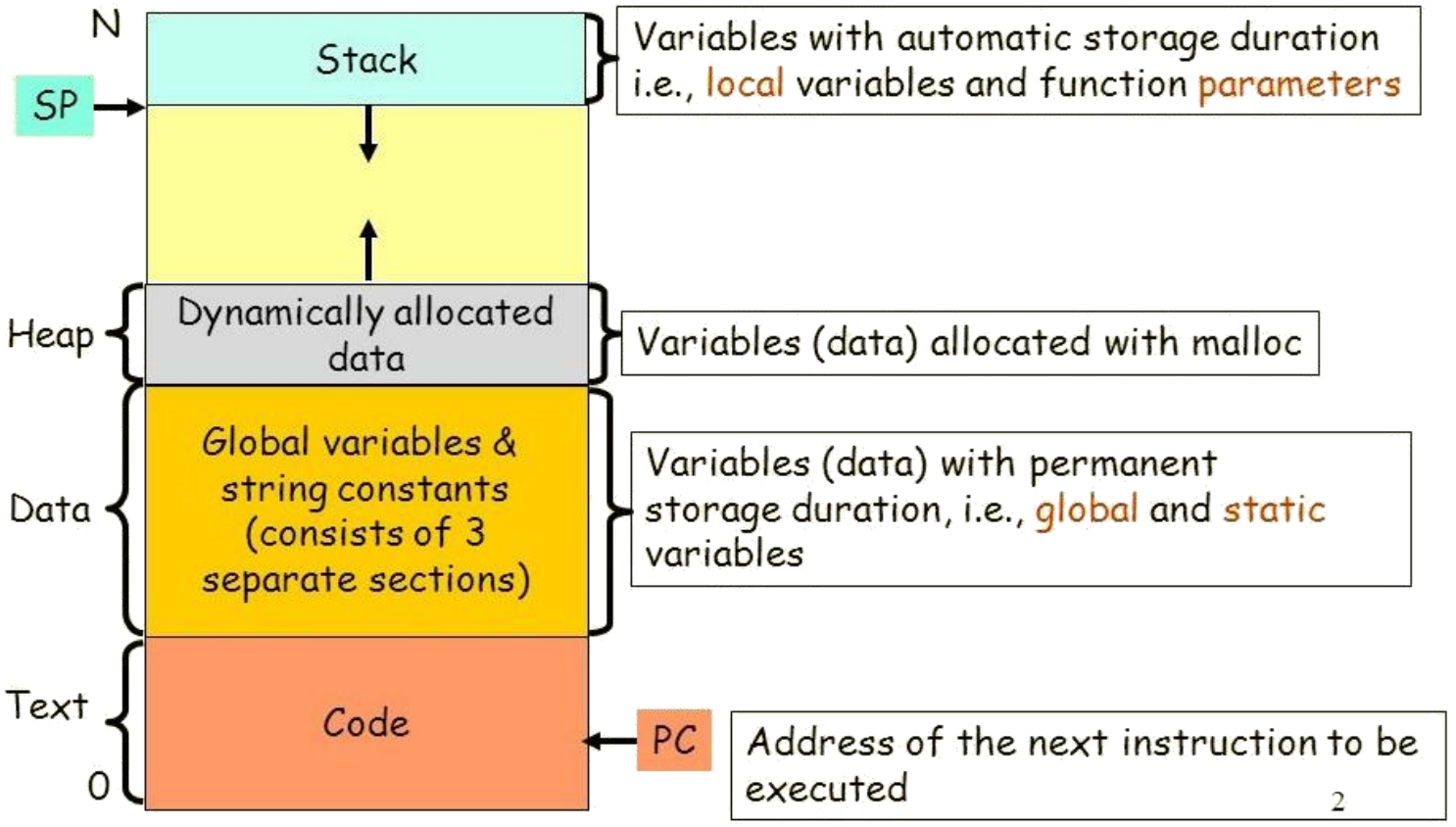
متغیر (Variable)

- در هر برنامه کامپیوتری، یک سری دیتای ورودی (یا اولیه) داریم که باید پردازش روی آن دیتا انجام شود.
- برای اینکه برنامه بتواند پردازش مورد نیاز را روی دیتا انجام دهد باید دیتا به **حافظه اصلی (موری)** انتقال داده شود.
- دیتای مورد نیاز برنامه باید در قالب **متغیر** داخل موری ذخیره گردد.



فضای حافظه کامپیوتر

(از زاویه دید برنامه نویس)



متغیر (Variable)

- ازآنجایی که در هر بار اجرای برنامه، آدرس قرارگیری دیتا در مموری تغییر می‌کند، به خاطرسپردن آدرس‌های حافظه و کار کردن با آنها برای برنامه‌نویس امکانپذیر نیست ☹

- به همین دلیل، هر متغیر دارای یک نام است که به محل ذخیره سازی دیتای آن متغیر در حافظه، اشاره دارد. آدرس معادل با نام متغیر در هر بار اجرای برنامه تغییر می‌کند ولی نیازی نیست برنامه‌نویس نگران این موضوع باشد. برنامه‌نویس فقط با نام متغیرها سر رو کار خواهد داشت ☺

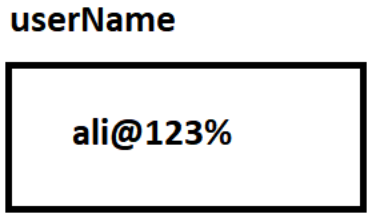
Address	Content	Name
90000000	00	sum
90000001	00	
90000002	00	
90000003	FF	
90000004	FF	age
90000005	FF	
90000006	1F	averge
90000007	FF	
90000008	FF	
90000009	FF	
9000000A	FF	
9000000B	FF	
9000000C	FF	
9000000D	FF	
9000000E	90	ptrSum
9000000F	00	
90000010	00	
90000011	00	

Note: All numbers in hexadecimal

متغیر (Variable)

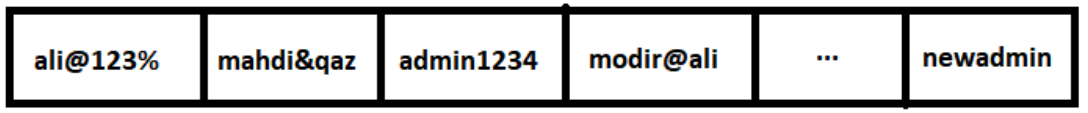
- متغیرها را می‌توان به دو نوع کلی تقسیم بندی کرد:

متغیرهای تک مقداری (Single Value)



- در این نوع متغیر، تنها یک دیتا را می‌توان ذخیره کرد.
- مثال: متغیر با دیتاتایپ string که برای نگهداری نام کاربری یک کاربر تعریف شده است. در این متغیر تک مقداری نمی‌توان نام کاربری چندین نفر را نگهداری کرد.

متغیرهای چندمقداری(Multiple Value)



- در این نوع متغیرها، می‌توان چندین مقدار (value) را نگهداری کرد.
- مانند متغیر آرایه‌ای از نوع string که قادر به ذخیره‌سازی نام کاربری چندین نفر می باشد.

متغیر (Variable)

- در نامگذاری متغیرها هر زبان قوانین خودش را دارد ولی پیشنهاد اکثر استانداردها این است که از a....z, A....Z , 0....9 استفاده نماییم و نام متغیر با عدد شروع نشود.
- از هر نام فقط یک بار می‌توان برای تعریف متغیر استفاده کرد و اسامی متغیرها نمی‌تواند تکراری باشد.
- زبان‌های مختلف قوانین متفاوتی در مورد متغیرها دارند.
  - برخی زبان‌ها مثل C تعریف متغیر قبل از استفاده از آن را اجباری کرده‌اند ولی برخی مانند python اجازه داده‌اند تا بدون نیاز به اعلان متغیر از آن استفاده شود.
  - در برخی زبان‌ها مانند C تعیین تایپ برای متغیر الزامی است ولی در برخی زبان‌ها مانند python نیازی به این کار نیست.

متغیر (Variable)

- **اعلان** (**Declaration**) : تعریف نوع و نام یک متغیر.

```
int a;
```

- **انتساب** (**Assign**) : مقداردهی یک متغیر.

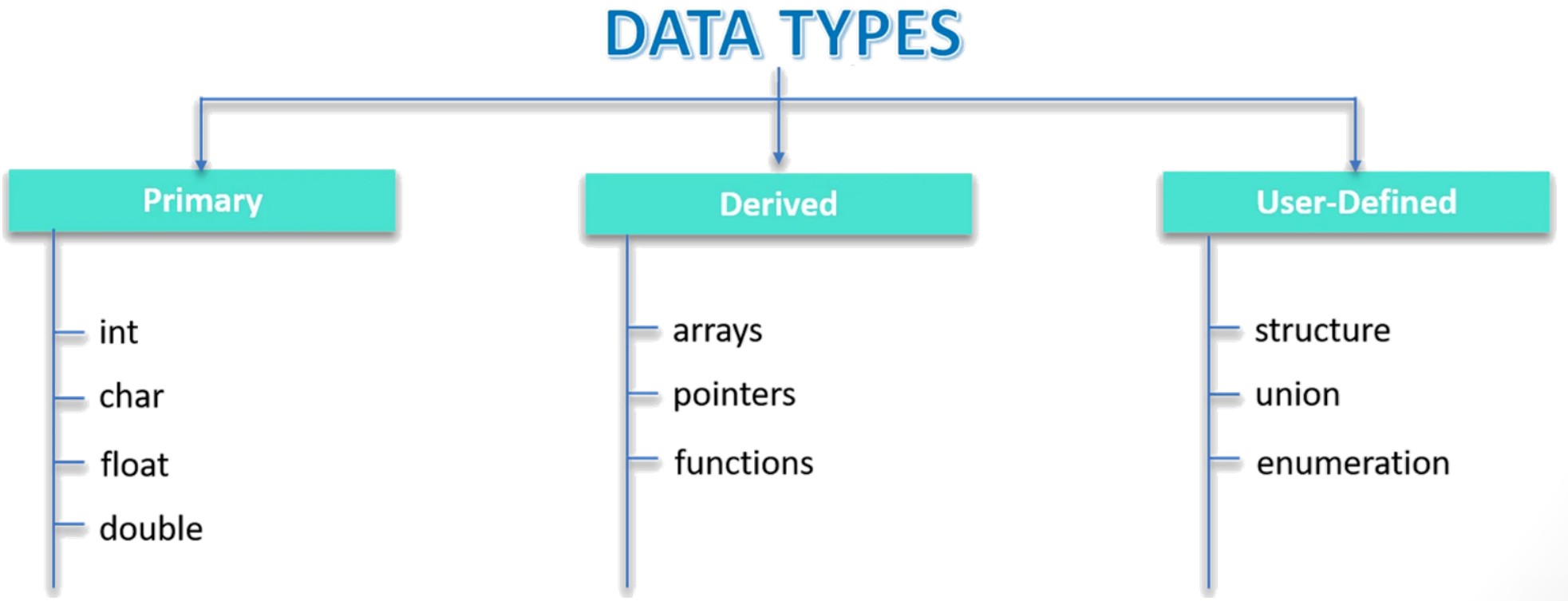
```
a = 10000;
```

- **دسترسی** یا **بازیابی** (Access / Retrieve) : خواندن مقدار یک متغیر.

```
Printf("Value of a = %d", a);
```

نوع داده (Data Types)

- هر متغیر باید **دیتاتایپ** (Data Type) مشخصی داشته باشد تا نوع دیتای ذخیره شده در آن محل حافظه، مشخص باشد.



دیتاتایپ‌های primitive

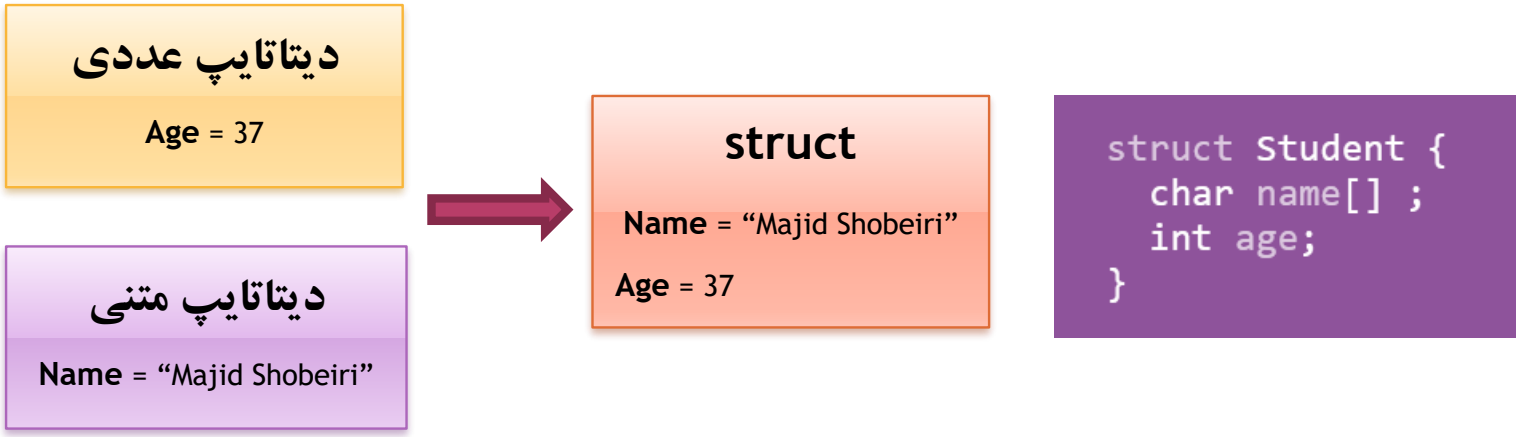
زبان‌های مختلف ممکن است برای یک تایپ مشخص مانند اعداد صحیح از کلمات کلیدی متفاوتی استفاده کرده باشند. ولی معمولاً نقاط اشتراک بالایی در این مورد بین زبان‌های برنامه‌نویسی دیده می‌شود.

دیتا تایپ‌های اولیه مشترک در زبان C و Java

Type	Keyword	Value range which can be represented by this data type
Character	char	-128 to 127 or 0 to 255
Number	int	-32,768 to 32,767 or -2,147,483,648 to 2,147,483,647
Small Number	short	-32,768 to 32,767
Long Number	long	-2,147,483,648 to 2,147,483,647
Decimal Number	float	1.2E-38 to 3.4E+38 till 6 decimal places

دیتاتایپ‌های user-defined

- برنامه‌نویس می‌تواند با ترکیب دیتا تایپ‌های اولیه (متنی، عددی، اعشاری و...) دیتا تایپ جدید (user-defined) ایجاد کند.
- برای مثال struct در زبان C چنین ویژگی دارد و برنامه‌نویس می‌تواند با توجه به نیاز، مجموعه‌ای از تایپ‌های اولیه مختلف را داخل یک تایپ جدید تحت عنوان struct جای دهد.



اعداد (Numbers)

- همه زبان‌های برنامه‌نویسی از انواع اعداد (صحیح و اعشاری) پشتیبانی می کنند.
- معمولاً دیتاتایپ‌های عددی زبان‌ها شامل تایپ‌های اصلی زیر می‌باشد.
- البته ممکن است یک زبان یک تایپ را نداشته باشد یا با کلمه کلیدی دیگری آن را مشخص کرده باشد و همچنین ممکن است تایپ های بیشتری برای اعداد داشته باشد.

Type	Keyword	Value range which can be represented by this data type
Number	int	-32,768 to 32,767 or -2,147,483,648 to 2,147,483,647
Small Number	short	-32,768 to 32,767
Long Number	long	-2,147,483,648 to 2,147,483,647
Decimal Number	float	1.2E-38 to 3.4E+38 till 6 decimal places

اعداد (Numbers)

- مثال از تعریف متغیر عددی

```
short  s;
int    i;
long   l;
float  f;
double d;

s = 10;
i = 1000;
l = 1000000;
f = 230.47;
d = 30949.374;

printf( "s: %d\n", s);
printf( "i: %d\n", i);
printf( "l: %ld\n", l);
printf( "f: %.3f\n", f);
printf( "d: %.3f\n", d);
```

Output:

```
s: 10
i: 1000
l: 1000000
f: 230.470
d: 30949.374
```

کاراکتر (char)

- هر یک از حروف الفبا (a, b, c, d...., A, B, C, D)، اعداد (0-9)، علائم نگارشی (” ; ’,) و علائم خاص (!@#$٪^&*)

اگر داخل علامت ’’ قرار گیرند، لیترال کاراکتری (character literal) یا ثابت کاراکتری (character constant) محسوب می‌شوند.

```
char ch = 'a’;
```

- هر متغیر از نوع **char** به اندازه **۸ بیت** از فضای حافظه را اشغال می کند.
- یعنی هر لیترال از ۲۵۶ کد اسکی که مقدار آن بین 127- تا 127 است را می‌توان در یک متغیر از نوع کاراکتر ذخیره کرد.
- توجه داشته باشید که متغیر از نوع char نمی‌تواند بیش از یک کاراکتر را نگهداری کند.

کاراکتر (char)

- مثال از تعریف کاراکتر

```
char  ch1;
char  ch2;
char  ch3;
char  ch4;

ch1 = ‘a’;
ch2 = ‘1’;
ch3 = ‘$’;
ch4 = ‘+’;

printf( “ch1: %c\n", ch1);
printf( “ch2: %c\n", ch2);
printf( “ch3: %c\n", ch3);
printf( “ch4: %c\n", ch4);
```

Output:

```
ch1: a
ch2: 1
ch3: $
ch4: +
```

کاراکترهای کنترلی (Escape Sequence)

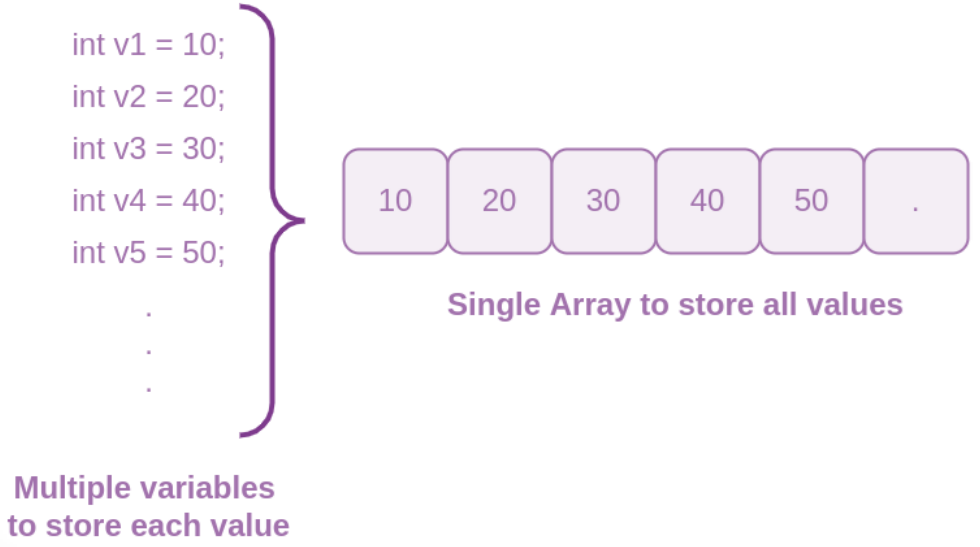
- وقتی یک کاراکتر بعد از علامت backslash (\) قرار گیرد، به آن **کاراکتر کنترلی** گفته می شود.
- **کاراکتر کنترلی** حاوی مفهوم خاصی برای کامپایلر می باشد (یک کاراکتر عادی نمایشی نیست).
- برای مثال کاراکتر کنترلی \n مفهوم new Line را به کامپایلر انتقال می دهد تا در محل قرار گیری این کاراکتر کنترلی، یک خط جدید در خروجی جاری ایجاد شود.

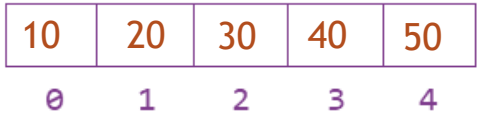
کاراکترهای کنترلی (Escape Sequence)

Escape Sequence	Description
\t	Inserts a tab in the text at this point.
\b	Inserts a backspace in the text at this point.
\n	Inserts a newline in the text at this point.
\r	Inserts a carriage return in the text at this point.
\f	Inserts a form feed in the text at this point.
\'	Inserts a single quote character in the text at this point.
\"	Inserts a double quote character in the text at this point.
\\	Inserts a backslash character in the text at this point.

آرایه (array)

- فرض کنید در یک برنامه لازم است شماره تماس افرادی که ثبت نام می‌کنند در حافظه نگداری شود.
- تعداد افرادی که ثبت نام می‌کنند ممکن است بسیار زیاد باشد یا حتی مشخص نباشد (متغیر باشد).
- بنابراین منطقی نیست که برای هر فرد، یک متغیر استفاده کنیم. در این شرایط از **آرایه (Array)** استفاده می‌کنیم.
- آرایه، یک دیتاتایپ drived چندمقداری است که از کنار هم قرار دادن چندین متغیر بدست می‌آید.
- تایپ همه خانه های آرایه یکسان بوده و نمی‌توان تایپ‌های مختلف (عدد، کاراکتر و ...) را در خانه‌های آرایه ذخیره کرد.

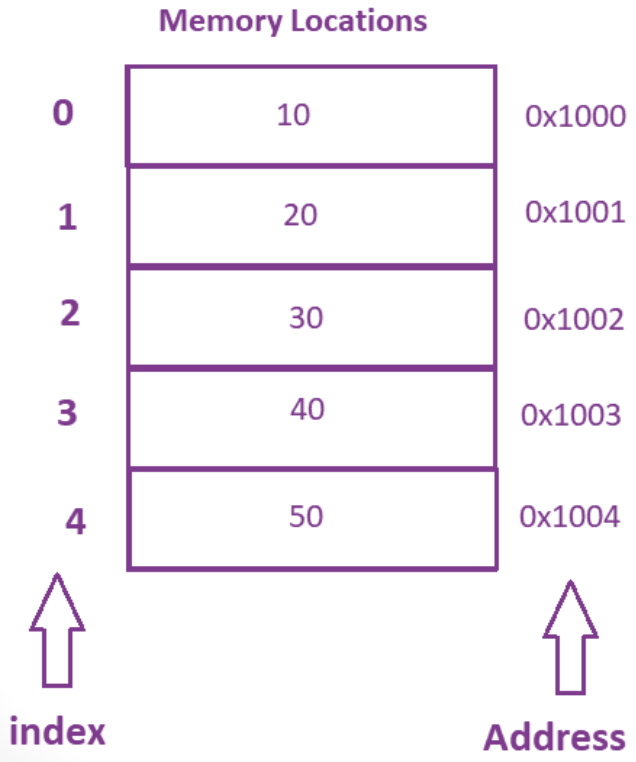


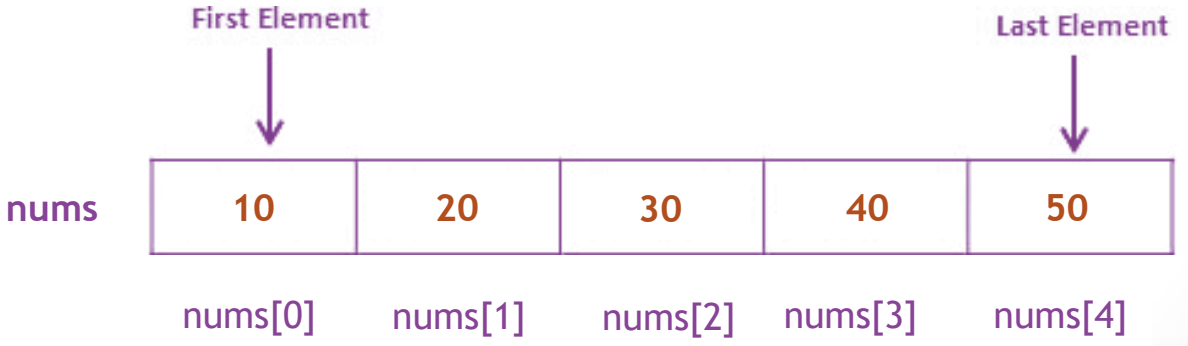



```
Size of the Array  = 5
Index of the Array = 0 1 2 3 4

First index = 0
Last index  = 4

First index is called lower bound
Last index is called upper bound
```

آرایه (array)

- آرایه یک **دیتاتایپ چندمقداری** است که به جای یک خانه، **چندین خانه متوالی از حافظه** (بسته به سایزی که برای آرایه مشخص می‌کنیم) را برای نگهداری مقادیر ما اختصاص می‌دهد.
- سایز هر خانه از آرایه در حافظه وابسته به تایپ انتخاب شده برای آرایه می‌باشد.
- پایین‌ترین آدرس، مربوط به عنصر اول آرایه و بالاترین آدرس، مربوط به عنصر آخر آرایه خواهد بود.





آرایه (array)

تعریف آرایه

```
type arrayName [ arraySize ];
int nums [ 5 ];
```

مقداردهی آرایه

```
int number[] = {10, 20, 30, 40, 50};
umber[4] = 50;
```

دسترسی به مقادیر آرایه

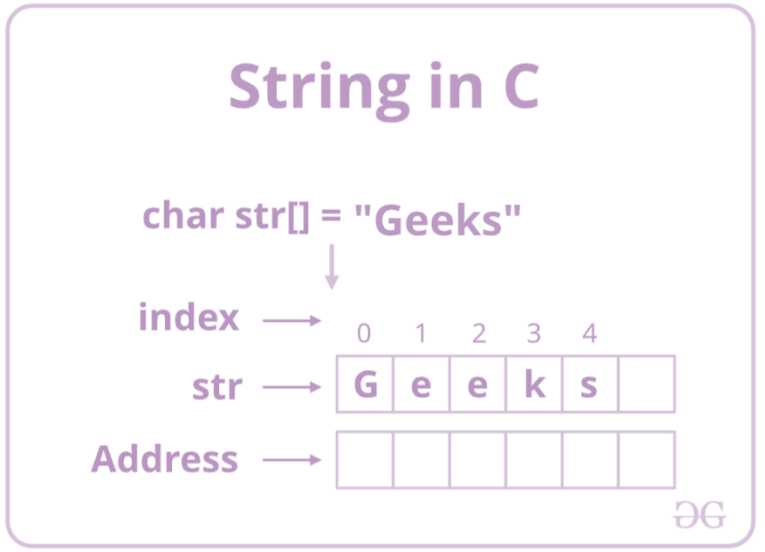
```
int var = number[4];
```

رشته (string)

- همانطور که گفته شد، نوع داده char تنها یک کاراکتر را در خود جای می‌دهد.
- حال اگر بخواهیم بیش از یک کاراکتر را در یک متغیر نگهداری کنیم، نیاز به دیتاتایپ string خواهیم داشت.

برخی زبان‌ها مانند زبان C دیتاتایپ رشته (string) را ندارند.

برنامه‌نویس باید به کمک **آرایه** و **char** (آرایه‌ای از کاراکترها) یک دیتاتایپ مناسب برای نگهداری متن بسازد.



رشته (string)

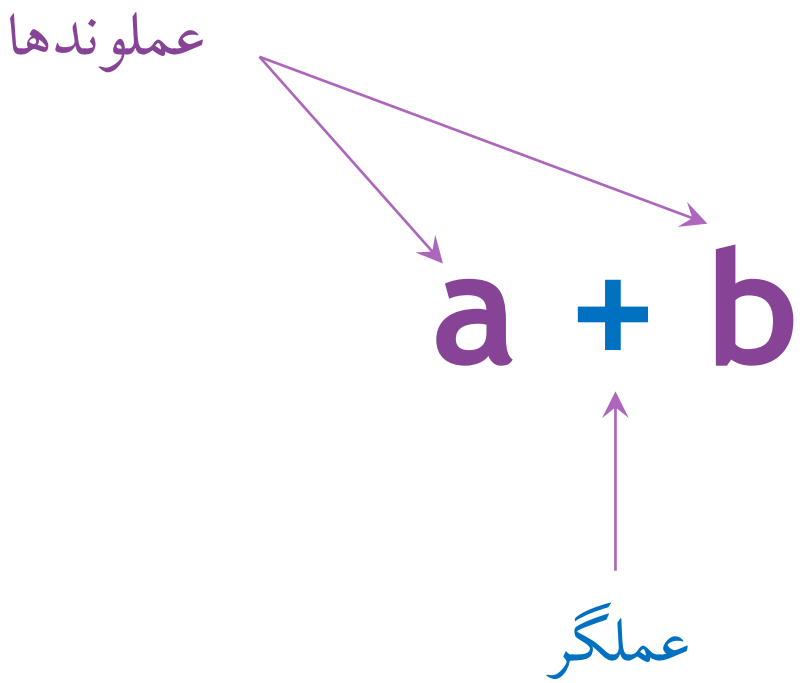
- توجه داشته باشید که رشته hello دارای ۵ کاراکتر است ولی ما آرایه ای به طول ۶ ایجاد می کنیم.
- وقتی رشته متنی را در آرایه ای از کاراکترها ذخیره می‌کنیم، کامپایلر یک **کاراکتر null** با کد \0 را در آخرین خانه آرایه قرار می‌دهد که نشان دهنده **پایان رشته** است.

```
char str[6] = “Hello”;
```

Code	Variable	Index	Value	Address
char str[6] = “Hello”;	str	0	H	1000
		1	e	1001
		2	l	1002
		3	l	1003
		4	o	1004
		5	\0	1005

عملگرها (Operators)

عملگر یا اپراتور در یک زبان برنامه‌نویسی، سمبولی است که به کامپایلر یا مفسر می‌گوید کدام عمل محاسباتی، منطقی یا رابطه‌ای را روی عملوندها انجام دهد.



عملگرهای محاسباتی در زبان C

Operator	Description	Example
+	Adds two operands	A + B will give 30
-	Subtracts second operand from the first	A - B will give -10
*	Multiplies both operands	A * B will give 200
/	Divides numerator by de-numerator	B / A will give 2
%	This gives remainder of an integer division	B % A will give 0

مثال : استفاده از عملگرهای محاسباتی

```
int a, b, c;

a = 10;
b = 20;
c = a + b;
c = a - b;
c = a * b;
c = b / a;
c = b % a;
```

عملگرهای مقایسه‌ای (رابطه‌ای) در زبان C

Operator	Description	Example
==	Checks if the values of two operands are equal or not, if yes then condition becomes true.	(A == B) is not true.
!=	Checks if the values of two operands are equal or not, if values are not equal then condition becomes true.	(A != B) is true.
>	Checks if the value of left operand is greater than the value of right operand, if yes then condition becomes true.	(A > B) is not true.
<	Checks if the value of left operand is less than the value of right operand, if yes then condition becomes true.	(A < B) is true.
>=	Checks if the value of left operand is greater than or equal to the value of right operand, if yes then condition becomes true.	(A >= B) is not true.
<=	Checks if the value of left operand is less than or equal to the value of right operand, if yes then condition becomes true.	(A <= B) is true.

عملگرها (Operators)

مثال : استفاده از عملگرهای مقایسه‌ای

```
if( a == 10 )
if( a != b )

if( a < b )
if( a > b )

if( a <= b )
if( a >= b )
```

عملگرهای منطقی در زبان C

Operator	Description	Example
&&	Called Logical AND operator. If both the operands are non-zero, then condition becomes true.	(A && B) is false.
||	Called Logical OR Operator. If any of the two operands is non-zero, then condition becomes true.	(A || B) is true.
!	Called Logical NOT Operator. Use to reverses the logical state of its operand. If a condition is true then Logical NOT operator will make false.	!(A && B) is true.

مثال : استفاده از عملگرهای منطقی

AND : `if( a && b )`

OR : `if( a || b )`

NOT : `if( ! a )`

جبر بولی : قوانین محاسبات منطقی

`! (a && b)  ~  (! a) || (! b)`

`! (a || b)  ~  (! a) && (! b)`

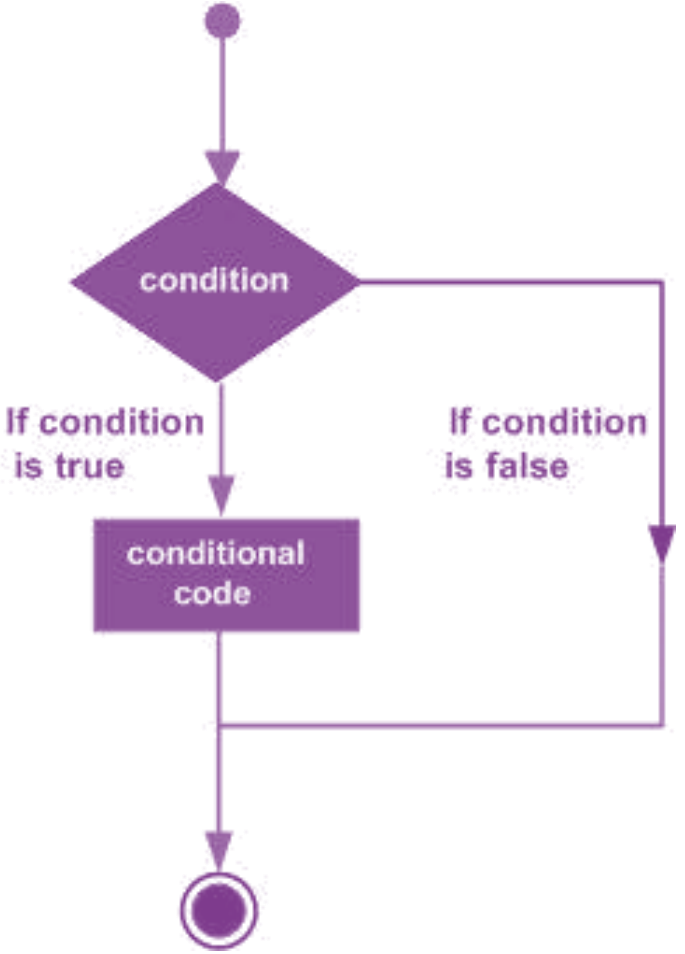
ساختار تصمیم (Decisions)

- در برخی شرایط لازم است بر اساس یک یا ترکیبی از چند شرط، یک گزینه از بین چند گزینه انتخاب شود.
- پیاده‌سازی موضوع تصمیم گیری در برنامه‌نویسی بسیار حیاتی بوده و تعیین شرط و کدنویسی ساختار تصمیم می بایست به دقت انجام شود.
- انواع ساختار تصمیم عبارتند از

  - if
  - if-else
  - if-elseif
  - ternary
  - switch

ساختار تصمیم (Decisions)

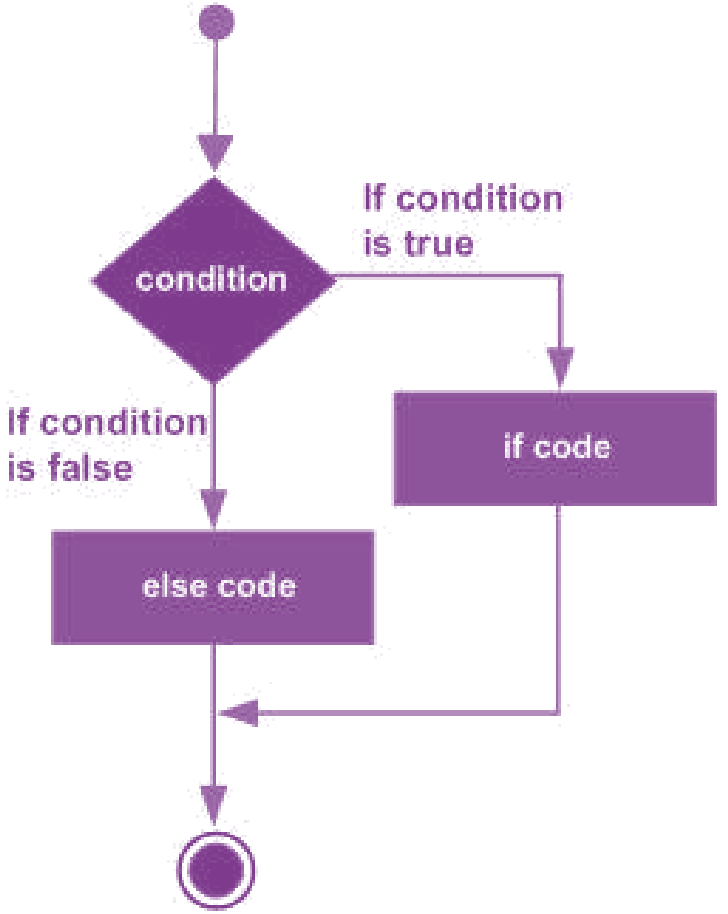
- پیاده‌سازی ساختار تصمیم با if




```
If (cond1) {
        option 1
}

If (cond2) {
        option 2
}
```

ساختار تصمیم (Decisions)

- پیاده‌سازی ساختار تصمیم با if-else

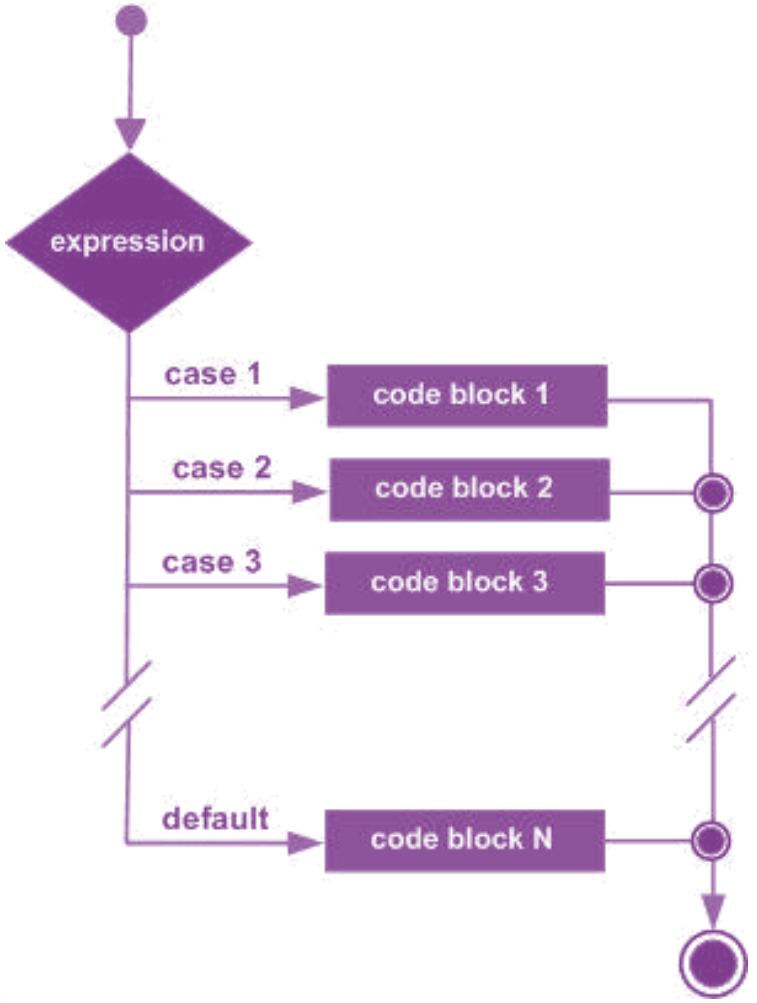



```
If (cond1) {
        option 1
}

else {
        option 2
}
```

ساختار تصمیم (Decisions)

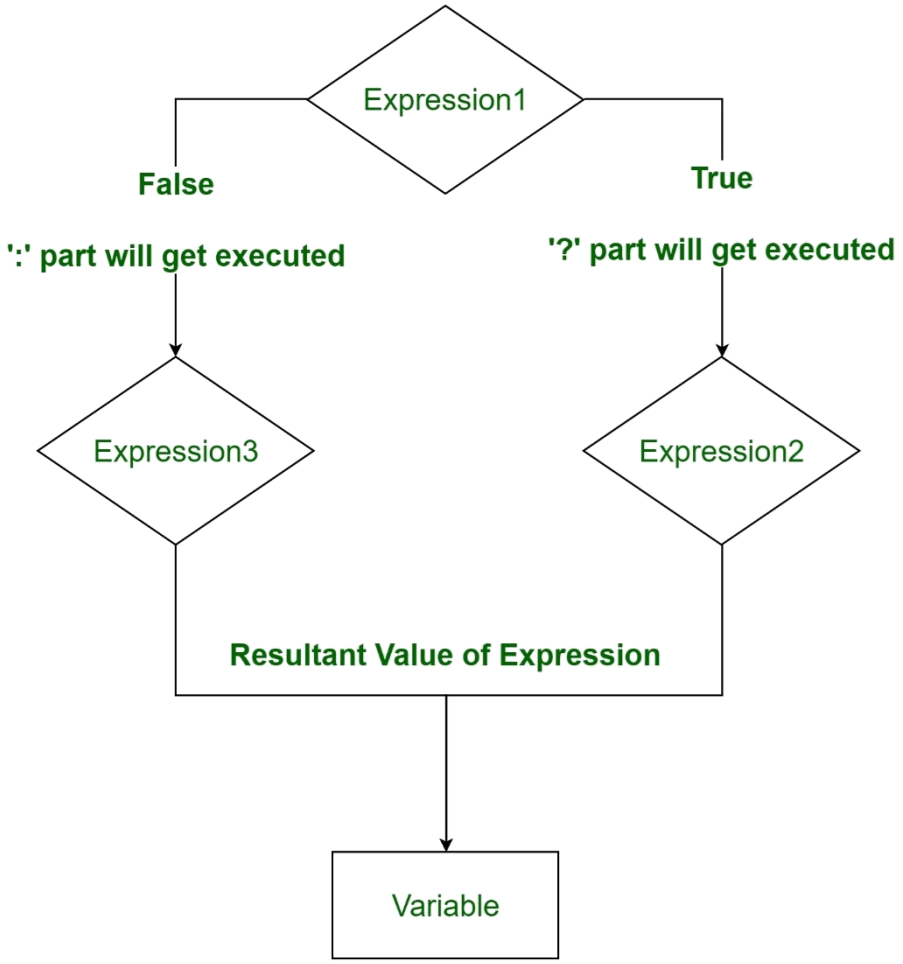
- پیاده‌سازی ساختار تصمیم با if-elseif-else

```
If (cond1) {
        option 1
}

else If (cond2) {
        option 2
}

else {
        option 3
}
```

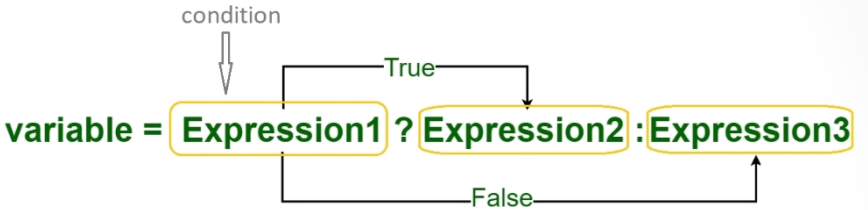


- پیاده‌سازی ساختار تصمیم با **switch-case**

```
switch( expression ){
    case 1 :
      printf( "One\n");
      break;
    case 2 :
      printf( "Two\n");
      break;
    case 3 :
      printf( "Three\n");
      break;
    case 4 :
      printf( "Four\n");
      break;
    default :
      printf( "None of the above...\n");
  }
```



پیاده‌سازی ساختار تصمیم با Ternary



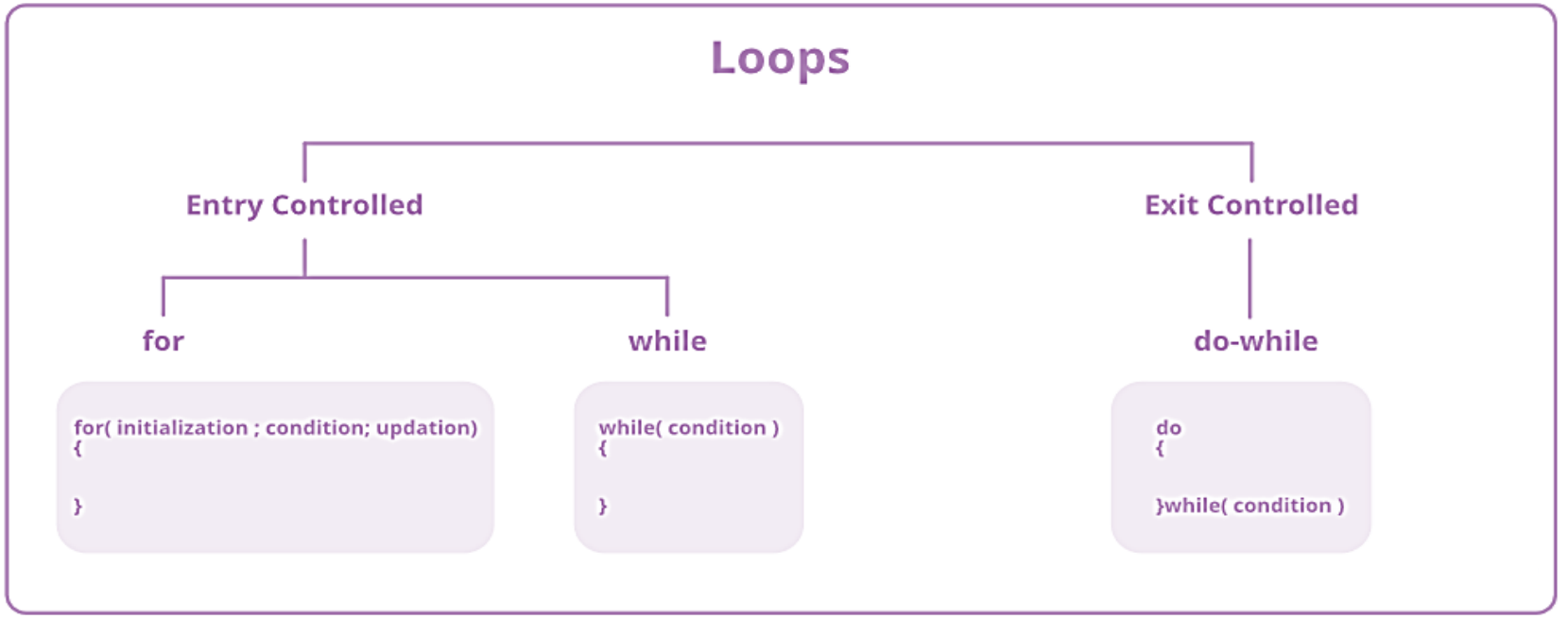

```
if(Expression1)
{
    variable = Expression2;
}
else
{
    variable = Expression3;
}
```

ساختار تکرار (Loop)

- فرض کنید تصمیم دارید پیامی را برای تعدادی از افراد ارسال کنید.
- ممکن است تعداد افراد زیاد باشد و یا حتی نامشخص و متغیر باشد.
- نوشتن دستور ارسال پیام برای تک تک افراد در این حالت، منطقی نخواهد بود.
- در چنین موقعیتی حلقه‌های تکرار راه حل مناسبی خواهند بود.
- یک حلقه تکرار می‌تواند دستورات مورد نظر ما را به تعداد مشخص (یا بر اساس مقدار یک متغیر) تکرار کند.

انواع ساختار تکرار

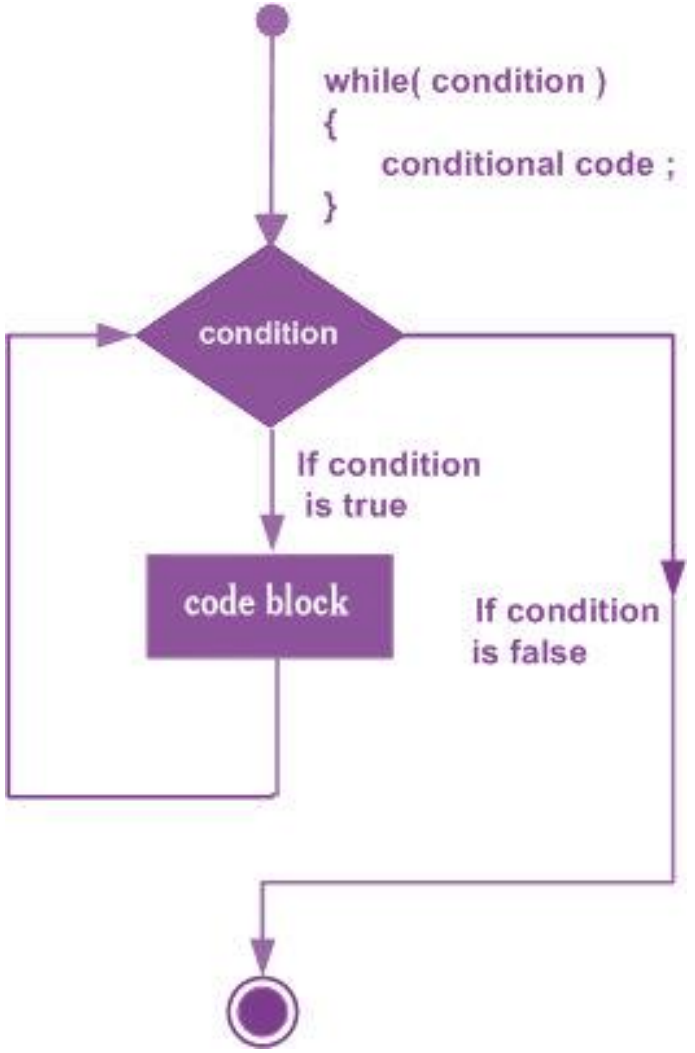
- while
- do-while
- for



ویژگی هر کدام از ساختارهای تکرار

1. **while loop** – First checks the condition, then executes the body.
2. **for loop** – firstly initializes, then, condition check, execute body, update.
3. **do-while** – firstly, execute the body then condition check

ساختار تکرار (Loop)



ساختار تکرار while

```
while (condition) {
        statement 1
        statement 2
        ...
        statement n
}

statement n+1
```

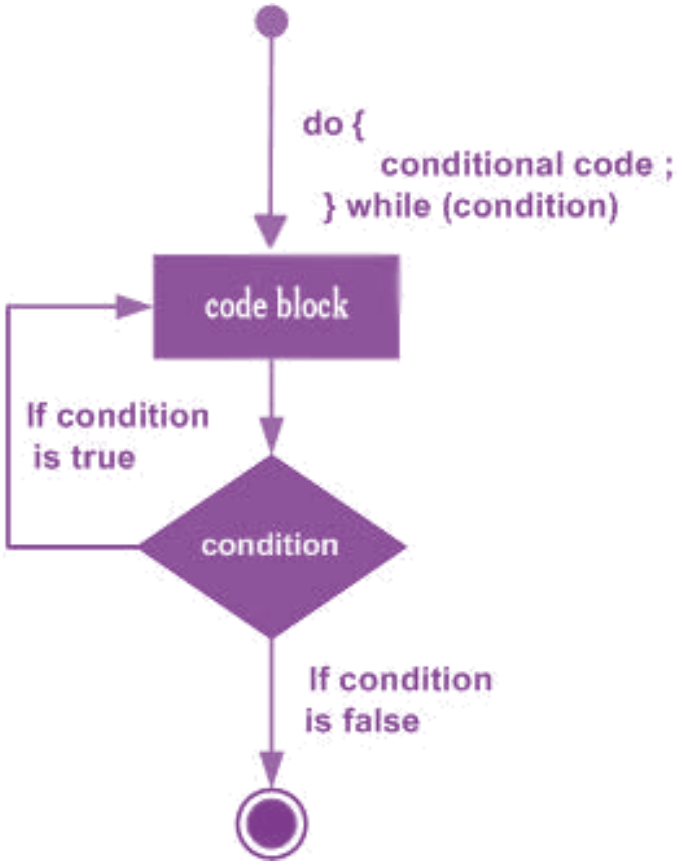
```
int i = 0;

while ( i < 5 ) {

    printf( "Hello!\n");

    i = i + 1;

}
```

ساختار تکرار (Loop)

- در ساختار تکرار while قبل از اجرای بدنه حلقه، ابتدا شرط حلقه چک می‌شود. اگر شرط صحیح بود وارد بدنه حلقه می‌شود در غیراینصورت اولین دستور بعد از بدنه حلقه اجرا خواهد شد.
- شرط حلقه معمولاً به صورت یک عبارت شرطی خواهد بود (مثال: a<= b) که نتیجه ارزیابی آن true یا false است. اگر نتیجه true باشد بدنه حلقه اجرا می‌شود و اگر false باشد اولین دستور بعد از بدنه حلقه اجرا خواهد شد.
- به جای عبارت شرطی، می‌توان از یک متغیر استفاده کرد. در اینصورت اگر مقدار متغیر غیرصفر باشد معادل true بوده و بدنه حلقه اجرا خواهد شد و اگر مقدار متغیر 0 باشد معادل false بوده و اولین دستور بعد از بدنه حلقه اجرا خواهد شد.
- اگر بدنه حلقه تکرار (همینطور بدنه if) تنها یک خط داشته باشد نیازی به {} نخواهد بود ولی اگر بیش از یک خط باشد، دستورات باید داخل {} قرار گیرد.

ساختار تکرار (Loop)

ساختار تکرار do-while



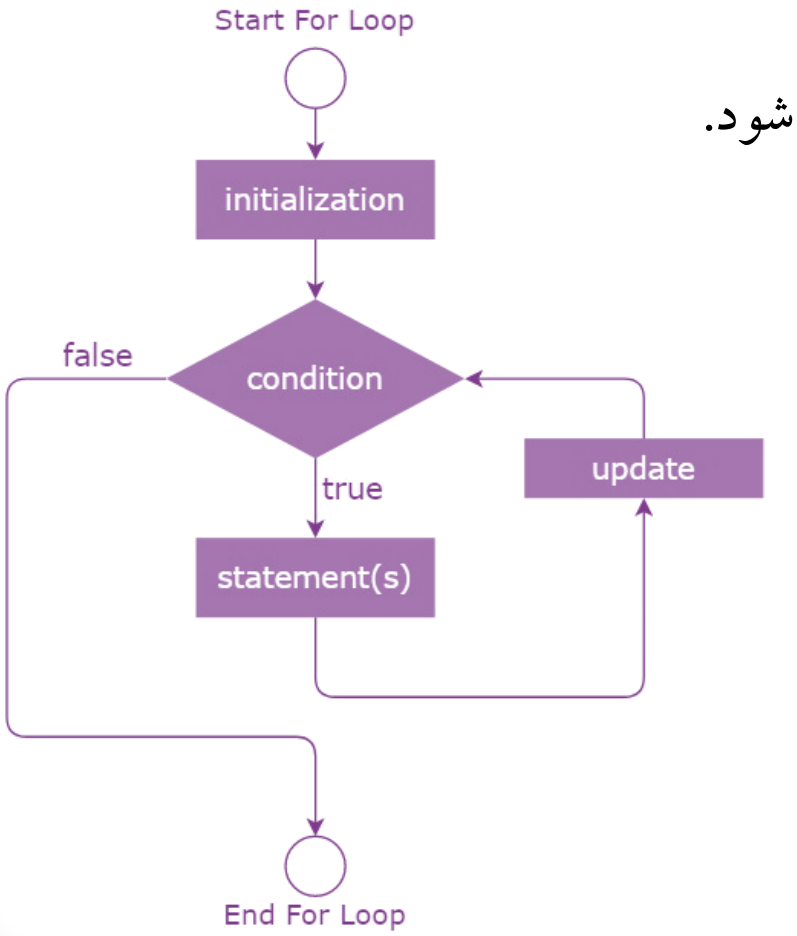

```
do {
        statement 1
        statement 2
        …
        statement n
}

While (condition);

statement n+1
```

```
int i = 0;

do {

    printf( "Hello!\n");

    i = i + 1;

}

while ( i < 5 )
```

ساختار تکرار (Loop)

- در ساختار تکرار do-while قبل از بررسی شرط حلقه، ابتدا یک دور بدنه حلقه اجرا می‌شود.
- در پایان اجرای دور اول، شرط حلقه چک می‌شود.
- اگر شرط صحیح بود وارد دور بعدی حلقه اجرا می‌شود در غیراینصورت اولین دستور بعد از حلقه اجرا خواهد شد.
- سایر ویژگی‌های این نوع حلقه دقیقاً مشابه حلقه while می‌باشد.

ساختار تکرار for



- در حلقه for ابتدا یک بار در آغاز کار، مقداردهی اولیه **اندیس حلقه** انجام می‌شود.
- سپس در هر بار اجرای حلقه، ابتدا شرط حلقه بررسی می‌شود.
- اگر شرط اجرای حلقه هنوز برقرار بود، دستورات بدنه حلقه اجرا می‌شود.
- اگر شرط برقرار نبود، حلقه به اتمام رسیده و دستور بعد حلقه اجرا می‌شود.
- پس از هر دور اجرای حلقه، مقدار شمارنده حلقه بروزرسانی می‌شود.

پیاده‌سازی حلقه for

```
for (initialization expr; test expr; update expr) {
    // statements we want to execute in loop
}
```

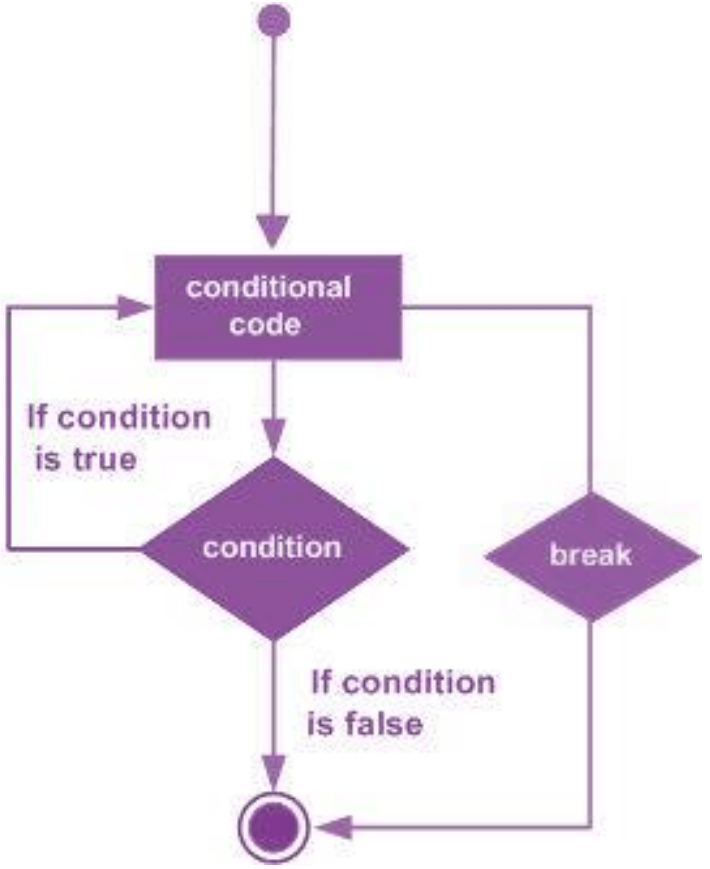
- بخش‌های تشکیل دهنده حلقه for
  - مقداردهی اولیه اندیس/شمارنده حلقه (int i=0)
  - شرط حلقه (i<5)
  - بروزرسانی اندیس با توجه به گام حلقه (i++)

* در این مثال، گام حلقه ۱ می باشد.

```
char str[6] = “Hello”;

for(int i = 0; i < 5; i++) {
    printf(“%c”, str[i]);
}
```

دستور break

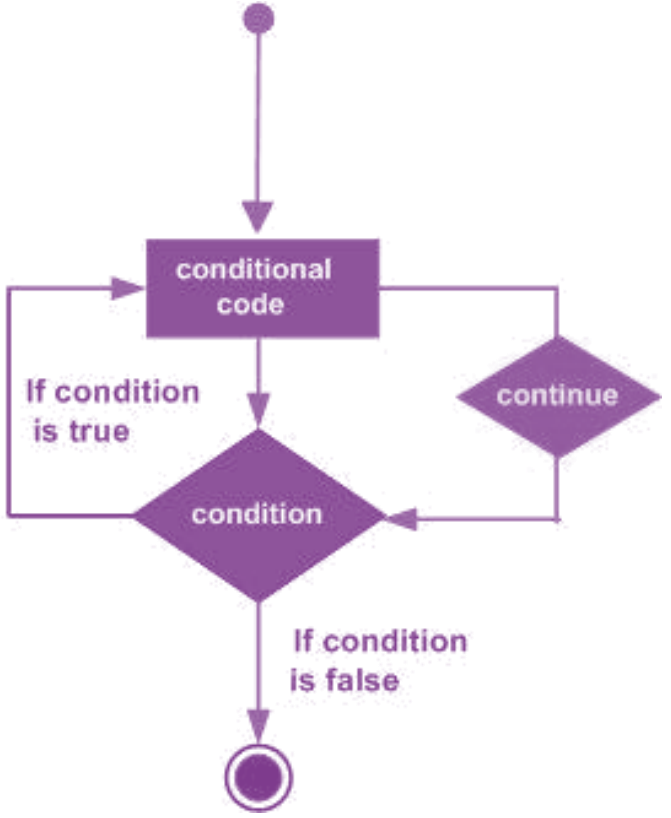
- هرگاه داخل بدنه حلقه تکرار از دستور break استفاده شود، اجرای حلقه در همان نقطه متوقف شده و اولین دستور بعد از حلقه اجرا خواهد شد.




```
int i = 0;
do {
      printf( "Hello!\n");
      i = i + 1;
      if(i==3)
            break;
}
while ( i < 5 )
```

دستور continue

- هرگاه **داخل بدنه حلقه** از دستور continue استفاده شود، اجرای **دور جاری** حلقه در همان نقطه متوقف شده (دستورات بعد از continue در دور جاری حلقه اجرا نمی‌شوند) و دور بعدی حلقه اجرا خواهد شد.



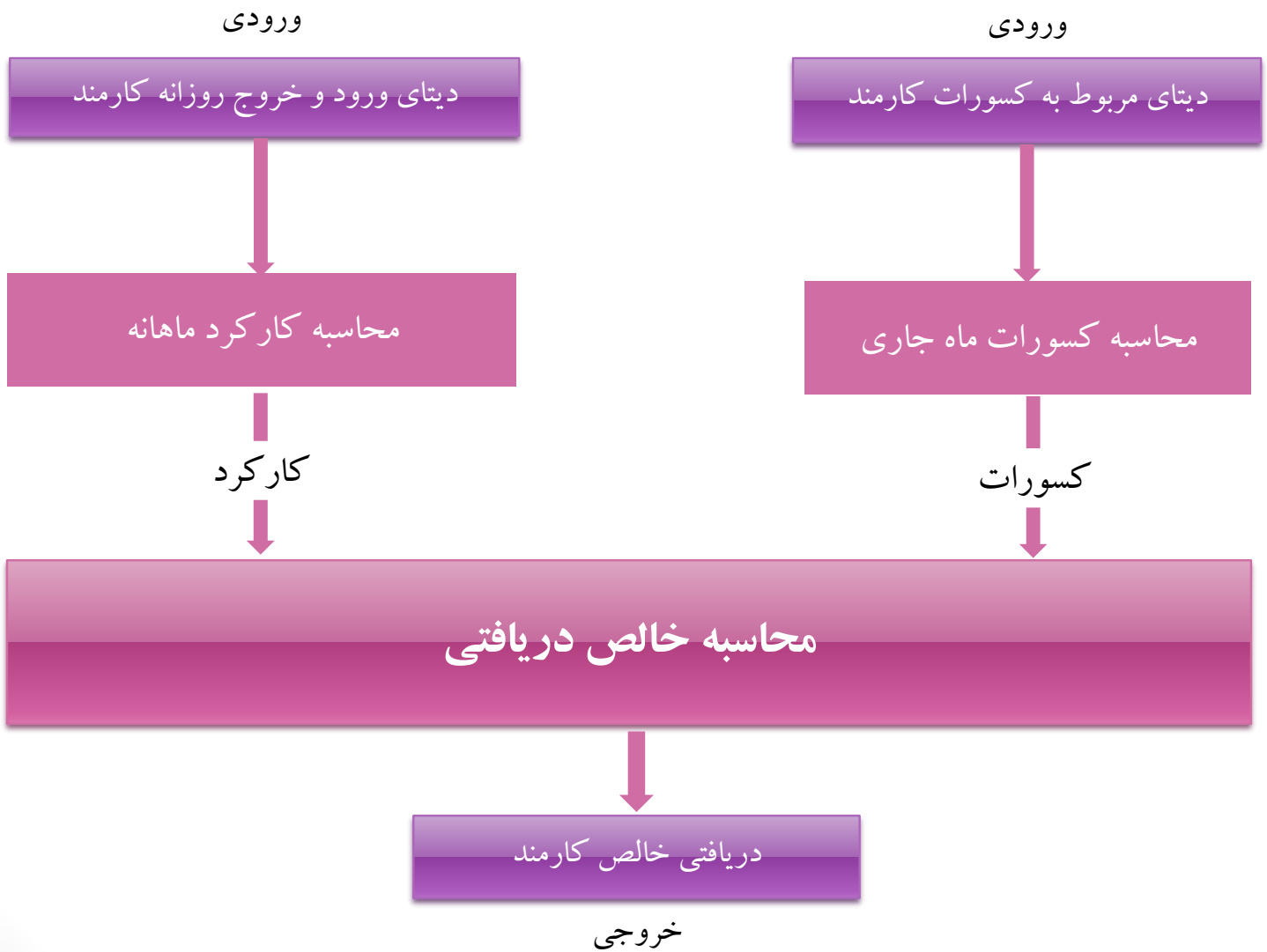

```
int i = 0;

do {
    if( i==3 ) {
        i = i + 1;
        continue;
    }
    printf( "Hello!\n");
    i = i + 1;
}
while ( i < 5 )
```

- **پارامترهای کدنویسی صحیح در اصول مهندسی نرم افزار**
  - **تمیز بودن** (Clean)
  - **خوانایی** (Readable)
  - **قابلیت استفاده مجدد** (Reusability)
  - **ماژولار بودن** (Modularity)
- برای رعایت این موارد در کدنویسی، ابتدا باید **وظایف (task)** در سیستم مورد نظر، جداسازی شود.
- هر وظیفه به صورت یک **سیستم پردازشی جداگانه** خواهد بود که یک سری **ورودی** را گرفته و پس از انجام محاسبات روی دیتای ورودی، **خروجی** مورد انتظار را ارائه می‌دهد.

مثال:

جداسازی Task های یک برنامه حسابداری

- **محاسبه کارکرد ماهانه**
- **محاسبه کسورات**
- **محاسبه خالص دریافتی**

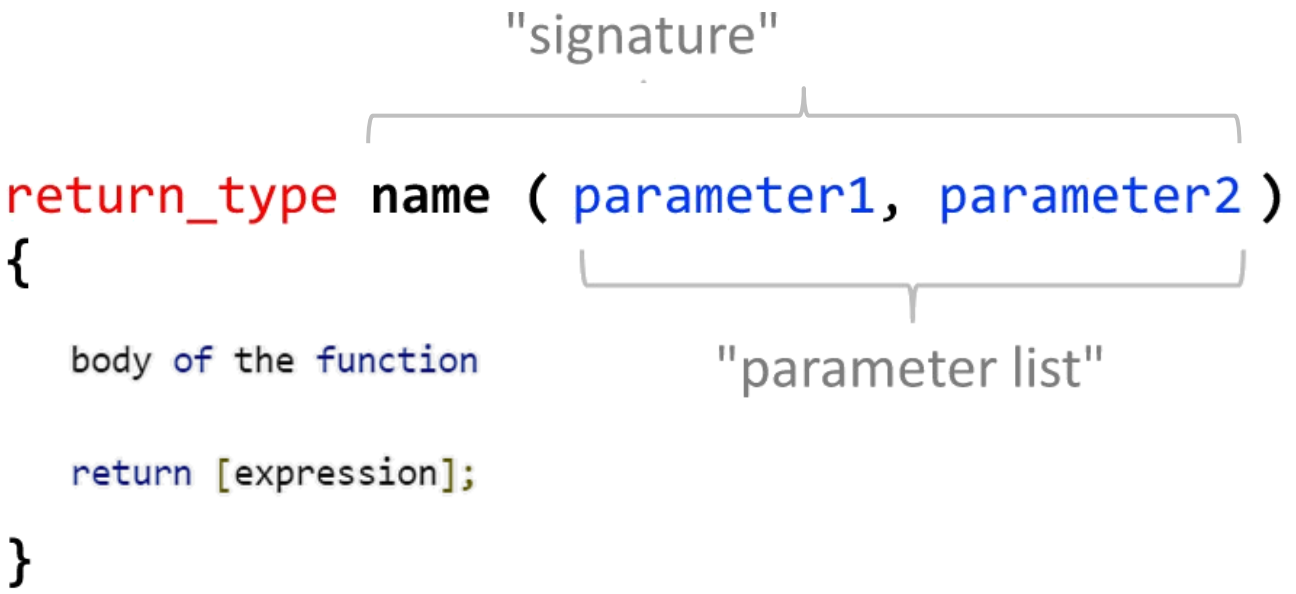


تابع (function)

- در یک زبان برنامه‌نویسی، هر وظیفه از سیستم، در قالب یک function پیاده‌سازی می‌شود.
- function عبارت است از چند خط کد که در قالب یک بلاک جداگانه، بسته‌بندی و نامگذاری شده است.
- هر فانکشن فقط یک بار پیاده‌سازی و پس از آن در مواقع نیاز، فراخوانی و استفاده خواهد شد.
- فانکشن‌های built-in فانکشن‌هایی هستند که در خود زبان وجود دارند. مانند فانشکن ()printf
- سایر فانکشن‌هایی که کاربر می‌نویسد، فانکشن‌های user-defined نامیده می‌شوند.
- زبان‌های مختلف ممکن است اسامی متفاوتی را برای فانکشن استفاده کنند ولی همه آنها مفهوم مشترکی دارند:

- functions
- methods
- sub-routine
- procedure

اجزاء function



- نام فانکشن (name)
- پارامترهای ورودی (parameter list)
- امضای فانکشن (signature)
- تایپ خروجی (return type)
- بدنه فانکشن (body)

- **پارامترهای ورودی** مانند **ظرف**هایی هستند که هنگام فراخوانی فانکشن، **مقادیر ورودی** در این ظرف‌ها قرار می‌گیرند.
- پارامترهای ورودی **اختیاری** هستند و ممکن است یک فانشکن هیچ پارامتر ورودی نداشته باشد.
- **ترتیب** و **تایپ** پارامترها در مقداردهی آنها هنگام **فراخوانی فانشکن** مهم است.
- در قسمت return type فانکشن، باید **دیتا تایپ خروجی** فانشکن درج شود.
  - **int** : فانکشنی که یک عدد صحیح برمی‌گرداند.
  - **void** : فانکشنی که هیچ مقدار خروجی نداشته باشد.

اجزاء تشکیل دهنده امضاء یک فانشکن

- **نام فانکشن**
- **تعداد پارامترها**
- **ترتیب پارامترها**
- **تایپ پارامترها**

- توجه داشته باشید که return type جزء امضاء فانکشن محسوب نمی‌شود.
- با یک نام یکسان می‌توان چندین فانشکن تعریف کرد (البته با امضاهای مختلف).
- هیچ دو فانکشنی نمی‌توانند امضاء یکسان داشته باشد.

مثال : تابع محاسبه مجموع دو عدد

```
int getSum(int a, int b) {
    int sum = a + b;
    return sum;
}
```

کلمات کلیدی (keywords)

- هر زبان برنامه‌نویسی دارای یک سری کلمات کلیدی است که برای آن زبان رزرو شده هستند.
- برنامه‌نویس نمی‌تواند از این اسامی برای نامگذاری متغیرها داخل برنامه استفاده کند.

کلمات کلیدی زبان C

auto	else	long	switch
break	enum	register	typedef
case	extern	return	union
char	float	short	unsigned
const	for	signed	void
continue	goto	sizeof	volatile
default	if	static	while
do	int	struct	_Packed
double			

مطالعه بیشتر

- https://www.tutorialspoint.com/computer_programming/index.htm
- https://www.tutorialspoint.com/learn_c_by_examples/index.htm
- https://www.tutorialspoint.com/cprogramming/index.htm
- https://www.tutorialspoint.com/java/index.htm
- https://www.tutorialspoint.com/python/index.htm
- https://www.tutorialspoint.com/compile_c_online.php
- https://www.learn-c.org/
- https://exercism.org/
- https://www.simplilearn.com/tutorials/c-tutorial/data-types-in-c
- https://www.simplilearn.com/tutorials/c-tutorial/conditional-control-statements-in-c
- https://www.geeksforgeeks.org/conditional-or-ternary-operator-in-c-c/

اصول و مبانی برنامه‌نویسی

مجید شبیری

کارشناسی ارشد IT، گرایش شبکه

از دانشگاه صنعتی امیرکبیر