درس «مبانی کامپیوتر و برنامه‌سازی»

ساختارهای شرطی و منطقی

صادق علی‌اکبری

سرفصل مطالب

- نکته جانبی: روش C++ برای ورودی و خروجی
  - cin و cout
- ساختارهای شرطی
  - ساختار if else
  - دستور switch
- ساختارهای شرطی تودرتو
- عملگرهای شرطی و مقایسه‌ای
- دستورات ترکیبی (blocks)

روش C++ برای ورودی و خروجی

cin & cout (console input & output)

- در زبان C++ دستورات جدیدی برای ورودی و خروجی ایجاد شده‌اند
  - دستور cin مشابه scanf
  - دستور cout مشابه printf
- استفاده از این دستورات ساده‌تر از printf و scanf است
  - اما این دستورات در زبان C وجود ندارند
- cin >> number;
  - یعنی مقدار متغیر number را از ورودی (کاربر) بخوان
- cout << number;
  - یعنی مقدار متغیر number را نمایش بده

مثال

- برنامه‌ای که یک عدد از ورودی می‌خواند و همان عدد را نمایش می‌دهد

```
#include <iostream>
using namespace std;

int main() {
  int number;

  cin>>number;
  cout<<number;

  return 0;
}
```

- به جهت >> و << دقت کنید
- به‌جای #include <stdio.h> می‌نویسیم:
- #include <iostream>
- using namespace std;

امکان چند ورودی یا خروجی در یک دستور

```
#include <iostream>
using namespace std;
int main() {
 int num1, num2;
 cin>>num1>>num2;
 cout<<"num1=" <<num1<<", num2=" <<num2 << '\n';
 cout<<"Finish!";

 return 0;
}
```

- نمونه اجرا:

```
4
7
num1=4, num2=7
Finish!
```

دستور if

دستور if

- یک شرط را بررسی می‌کند
- اگر شرط برقرار بود (true) دستور بعدی را اجرا می‌کند
- مثال:

```
unsigned int number;
cin>> number;
if( number<10 )
  cout<<"Tak Raghami!";
```

- دقت: در انتهای دستور if از سمیکالن استفاده نکنید

دستور else در کنار if

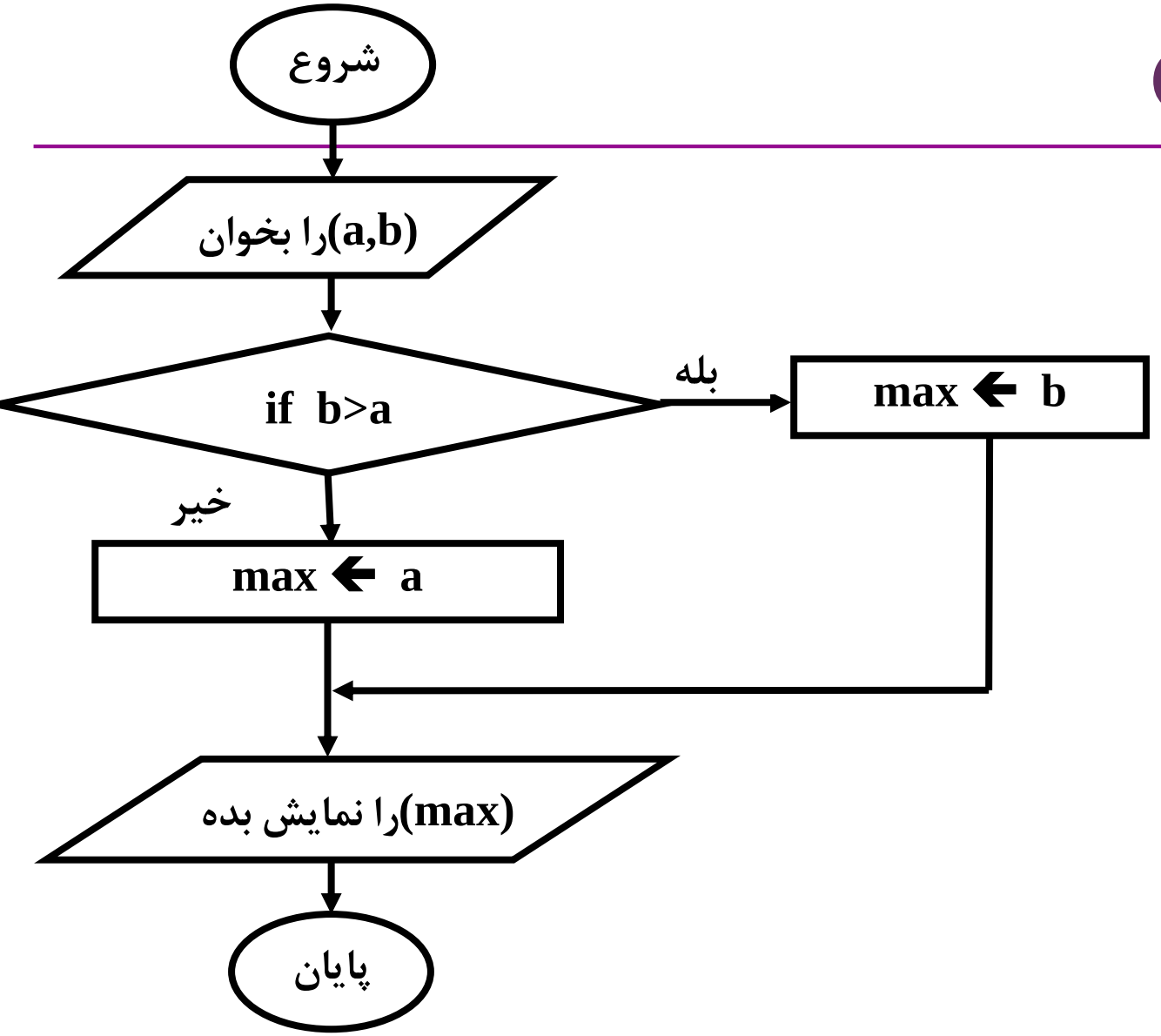
- اگر شرط برقرار نباشد، بخش else اجرا می‌شود

- مثال:

```
unsigned int number;
cin>> number;
if( number<10 )
  cout<<"Tak Raghami!";
else
  cout<<"Chand Raghami!";
```

- دقت: در انتهای دستور else هم از سمیکالن استفاده نکنید

تبدیل فلوچارت به برنامه C




```
#include <iostream>
using namespace std;
int main()
{
 int a,b, max;
 cin>> a>>b;
 if(b>a)
    max = b;
 else
    max = a;
 cout<<max;
 return 0;
}
```

تمرین

- برنامه‌ای بنویسید که سه عدد از ورودی بگیرد و کمترین آن‌ها را چاپ کند
  - از دستورات ترکیبی یا شرط‌های تودرتو (که بعداً یاد می‌گیریم) استفاده نکنید

```
#include <iostream>
using namespace std;
int main() {
 int num1, num2, num3;
 cin >> num1 >> num2 >> num3;
 int min = num1;
 if(min>num2)
   min = num2;
 if(min>num3)
   min = num3;
 cout << min;
 return 0;
}
```

مقدار منطقی

- آن‌چه داخل پرانتز به دستور if داده می‌شود، یک مقدار منطقی است
  - Boolean value
  - یکی از دو مقدار true یا false
- مثال:

```
if(false)
```

```
if(true)
```

```
if(b>max)
```

```
bool b_gt_max = b > max;
if (b_gt_max)
```

- یک مقدار منطقی معمولاً با کمک عملگرهای منطقی (مثل عملگر >) حاصل می‌شود

یادآوری: نوع داده منطقی (boolean)

- نوع داده bool
- یک مقدار true یا false نگه می‌دارد

```
bool condition;
condition = false;
condition = true;
condition = a>b ;
```

- یکی از کاربردها: در دستورات شرطی (if...else)
  - آن چه در پرانتز به if داده می‌شود، از نوع bool است

عملیات ترکیبی (compound statement)

- نکته:
  - اگر شرط if برقرار باشد، فقط اولین دستور بعد از if اجرا می‌شود
  - اگر شرط if برقرار نباشد، فقط اولین دستور بعد از else اجرا می‌شود
- اگر بخواهیم در هر یکی از این شرایط چند دستور را اجرا کنیم چه؟
- مثال:
  - برنامه‌ای که دو عدد از ورودی بگیرد و آن‌دو را به ترتیب نزولی مرتب کند (sort)
- در این موارد از آکولاد باز و آکولاد بسته برای ایجاد یک بلوک (block) استفاده می‌کنیم

مثال

- برنامه‌ای که دو عدد از ورودی بگیرد و آن‌دو را به ترتیب نزولی مرتب کند (sort)

```
int num1, num2;
cin >> num1 >> num2;
if (num1 < num2)
{
  int temp = num1;
  num1 = num2;
  num2 = temp;
}
cout<<num1<<'\n'<<num2;
```

بلوک
Block

- اگر آکولادها را نمی‌گذاشتیم چه می‌شد؟

تمرین

- برنامه‌ای بنویسید که سه عدد از ورودی بگیرد و آن‌ها را مرتب و سپس چاپ کند (صعودی)

نکته: دندانه‌گذاری (indentation)

```
#include <iostream>
using namespace std;
int main() {
	int num1, num2, min;
	cin >> num1 >> num2 ;
	if(num1>num2)
	{
		min = num2;
		cout<<"num1>num2 \n";
	}
	else
	{
		min = num1;
		cout<<"num2>=num1 \n";
	}
	cout<<min;
	return 0;
}
```

- اجبار زبان نیست
- از نظر نحوی وجود یا عدم وجود آن هیچ فرقی ندارد

- یک عرف برنامه‌نویسی است
- خواندن و فهمیدن برنامه را ساده‌تر می‌کند

عملگرهای مقایسه‌ای

عملگرهای مقایسه‌ای

Operator	Significance
<	less than
<=	less than or equal to
>	greater than
>=	geater than or equal to
==	equal
!=	unequal

- توجه: تفاوت عملگرهای = و ==

مثال

Comparison	Result
`5 >= 6`	false
`1.7 < 1.8`	true
`4 + 2 == 5`	false
`2 * 4 != 7`	true

مثال

```
if(num1<num2)
  cout<<"num1<num2 \n";
else if(num1==num2)
  cout<<"num1=num2 \n";
else
  cout<<"num1>num2 \n";
```

```
if(num1 <= num2)
```

```
if(num1 >= 5)
```

```
if(num1 != 5)
```

عملگرهای منطقی

عملگرهای منطقی

- Logical Operators یا Logical connective
  - عملگرهایی که بر روی یک یا چند عملوند منطقی (boolean) اجرا می‌شوند
    - خروجی آن‌ها هم یک مقدار منطقی است (true یا false)
  - عملگر منطقی دوعملوندی AND که با && نمایش داده می‌شود
    - اگر هر دو عملوند صحیح (true) باشند، خروجی صحیح خواهد بود
  - عملگر منطقی دوعملوندی OR که با || نمایش داده می‌شود
    - اگر حداقل یک عملوند صحیح (true) باشد، خروجی صحیح خواهد بود
  - عملگر منطقی تک‌عملوندی NOT که با ! نمایش داده می‌شود (نقیض عملوند را برمی‌گرداند)

عملگرهای منطقی

A	B	A && B	A || B
true	true	true	true
true	false	false	true
false	true	false	true
false	false	false	false

A	!A
true	false
false	true

مثال

فرض
x = true
y = false
a = 5
b = 7
PI = 3.14

مثال	نتیجه
x && y	FALSE
!x || !y	TRUE
x == y	FALSE
a>b || PI<a	TRUE
!y && a<b && PI != 3	TRUE

نکته

- مقادیر منطقی (boolean) در واقع به صورت عدد صحیح (int) ارزیابی می‌شوند
  - مقدار true معادل یک و false معادل صفر است
- در ارزیابی یک عبارت منطقی، مقدار صفر معادل false در نظر گرفته می‌شود
- هر مقدار غیرصفر هم معادل true در نظر گرفته می‌شود
- ولی:
  - بهتر است از مقدار منطقی به عنوان عدد استفاده نکنیم
  - بهتر است از عدد به عنوان مقدار منطقی استفاده نکنیم

مثال

فرض
x = true
a = 5
b = 7
PI = 3.14

اما هیچ یک از موارد زیر الگوی مناسبی برای استفاده از مقادیر منطقی نیست

مثال	نتیجه
x && a	TRUE (1)
PI || b	TRUE (1)
true == 1	TRUE (1)
0 == false	TRUE (1)
5 && -1 && 3 && 3.14 && 1765	TRUE (1)
5 && -1 && 3 && 3.14 && 1765 && 0	FALSE (0)

سؤال

- خروجی قطعه‌برنامه زیر چیست؟ چرا؟
  - دقت کنید برای مقایسه، از = به جای == استفاده شده است
- پاسخ صحیح: YES چاپ می‌شود
- دلیل:
  - عبارت a=6 دو کار انجام می‌دهد:
    ۱- مقدار a را برابر با ۶ قرار می‌دهد
    ۲- مقدار ۶ را برمی‌گرداند
  - می‌دانیم هر مقدار غیرصفر (مثلاً ۶) در شرط به عنوان true در نظر گرفته می‌شود

```
int a = 5;
if(a = 6)
    cout<<"YES";
else
    cout<<"NO";
```

تمرین

- برنامه‌ای بنویسید که سه عدد از ورودی بگیرد و کمترین آن‌ها را چاپ کند

```
int a,b,c;
cin >>a >> b >> c;
if(a<b)
  if(a<c)
    cout<<a;
  else
    cout<<c;
else
  if(b<c)
    cout<<b;
  else
    cout<<c;
```

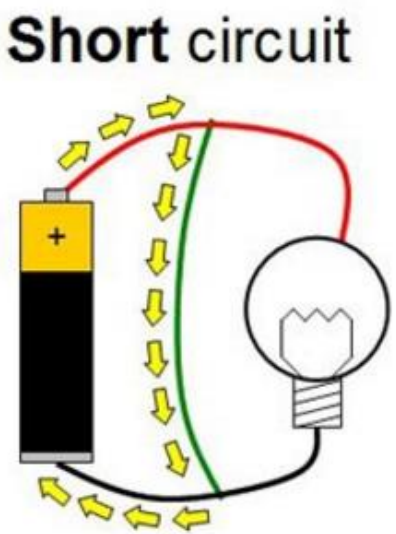
شرط‌های تودرتو
(Nested)

```
int a,b,c;
cin >>a >> b >> c;
if(a<b && a<c)
  cout<<a;
else if(b<c)
  cout<<b;
else
  cout<<c;
```

مفهوم اتصال کوتاه

Short-circuit evaluation

- در ارزیابی (evaluation) یک عبارت منطقی، گاهی ارزیابی کل عبارت لازم نیست
- مثال: x==x || x>5 چگونه ارزیابی خواهد شد؟
  - عبارت فوق true است و لازم نیست بخش x>5 ارزیابی شود
  - زیرا در ارزیابی OR ، صحیح (true) بودن یکی از عملوندها برای تعیین نتیجه کافیست



- به همین ترتیب در عملگر AND
  - اگر یکی از عملوندها false باشد، نتیجه مشخص است (false)
  - لازم نیست عملوندهای بعدی ارزیابی شوند
- در این موارد C و C++ تا جایی که لازم است ارزیابی را ادامه می‌دهند

مثال برای اتصال کوتاه

```
int b ;
int a = 0;
cin>>b;
if (a == 0 || b==0)
{
    cout<<"ZERO";
}
```

ZERO

```
char ch = 'A';
if (false && ch == 'A')
{
  cout << "YES";
} else {
  cout << "NO";
}
```

NO

```
int a = 0;
int b = 5;
if (a < 0 || a>=0 || ++b>0)
{
    cout<<"OK\n";
}
cout<<b;
```

OK
5

شرط‌های تودرتو (Nested Conditions)

- شرطی که درون یک شرط دیگر قرار گرفته باشد
- یعنی یک if که به عنوان بدنه یک if یا یک else به‌کار گرفته شود
- رعایت دندانه‌گذاری مناسب

```
if(condition1)
{
    statement1;
    if(condition2)
    {
        statement2;
    }
    else{
        statement3;
    }
}
```

```
if(condition1)
{
statement1;
if(condition2)
{
statement2;
}
else{
statement3;
}
}
```

یادآوری: عملگر سه‌عملوندی شرطی

- نحوه استفاده:

```
TYPE value = CONDITION ? Val_True : Val_False;
```

- مثال:

```
double area = 2 * 2 * 3.14;
int value = area > 10 ? 1 : -1;
//value = 1;
```

```
if(area>10)
    value = 1;
else
    value = -1;
```

دستور switch

ساختار Switch

```
switch(x){
case 1: y++; break;
case 2: y--; break;
case 3: z=5; break;
}
```

- کاربرد: وقتی که شرط‌ها، بررسی تساوی یک مقدار با گزینه‌های مختلف است
  - اگر x برابر با a بود فلان‌کار را بکن، اگر x مساوی b بود بهمان‌کار را بکن و ...
  - مثال فوق: اگر x==1 مقدار آن‌گاه y++ وگرنه اگر x==2 آن‌گاه y--، و اگر x==3 آن‌گاه z=5
- جایگزینی برای ساختار if-else با ساختمانی بهتر
  - خوانایی برنامه بهتر می‌شود
- آیا هر دستور if را می‌توانیم با یک switch-case بازنویسی کنیم؟
  - خیر

مرور ساختار switch

```
Switch ( expression )
{
   case const1:   [ statement ; ]
                  [ break ; ]
   case const2:   [ statement ; ]
                  [ break ; ]
        .
        .
        .
   [ default :    statement ]
}
```

default یعنی: اگر هیچ کدام نبود

یعنی مقدار expression با هیچ‌یک از case ها برابر نباشد

مثالی برای Switch

```
switch (i) {
case 1:
        cout << "YES";
      break;
case 2:
        cout << "NO";
      break;
default:
        cout << "Perhaps";
}
```

معادل همین switch-case با کمک if-else:

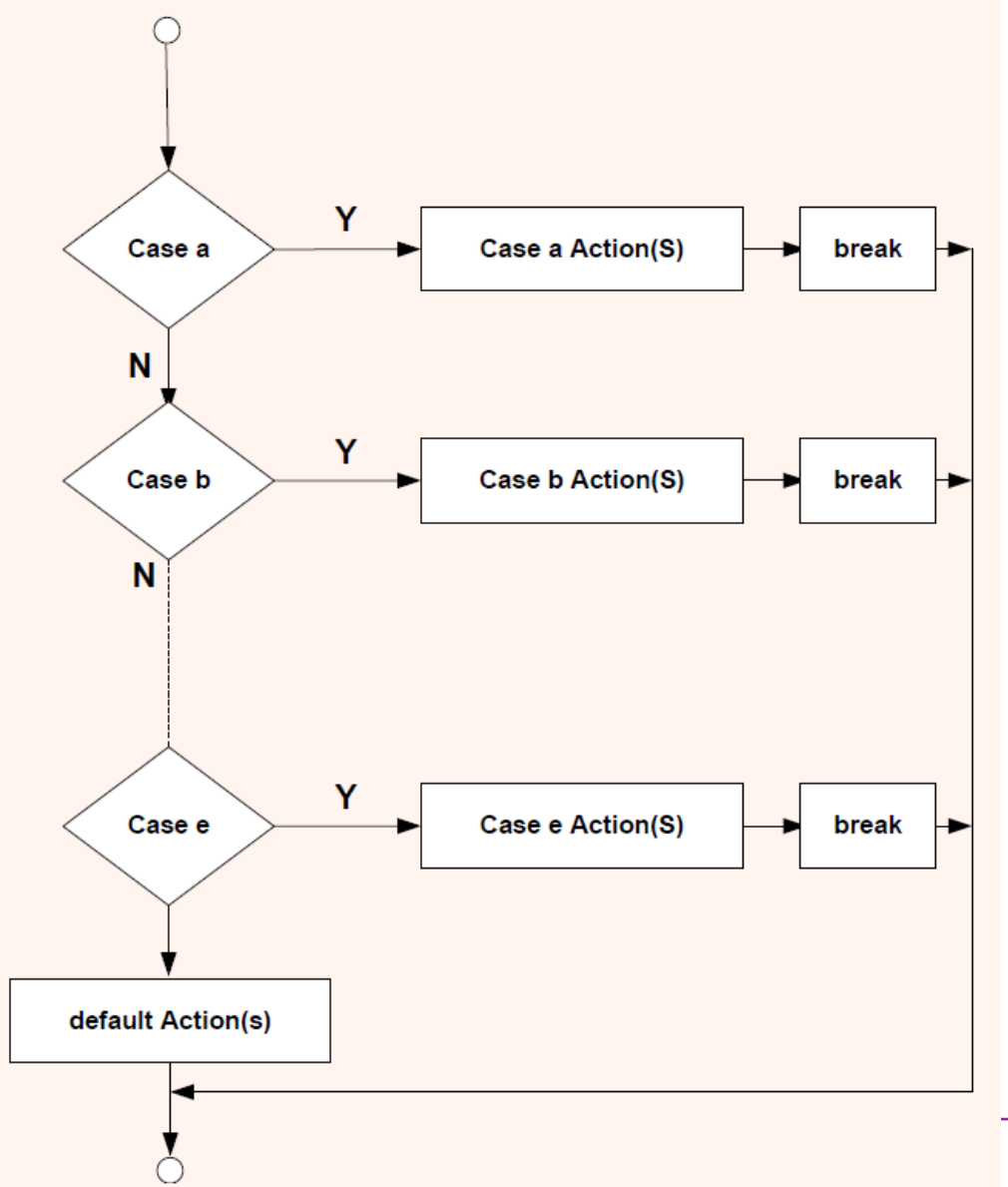
```
if(i==1)
   cout<<"Yes";
else if(i==2)
   cout<<"No";
else
   cout<<"Perhaps";
```

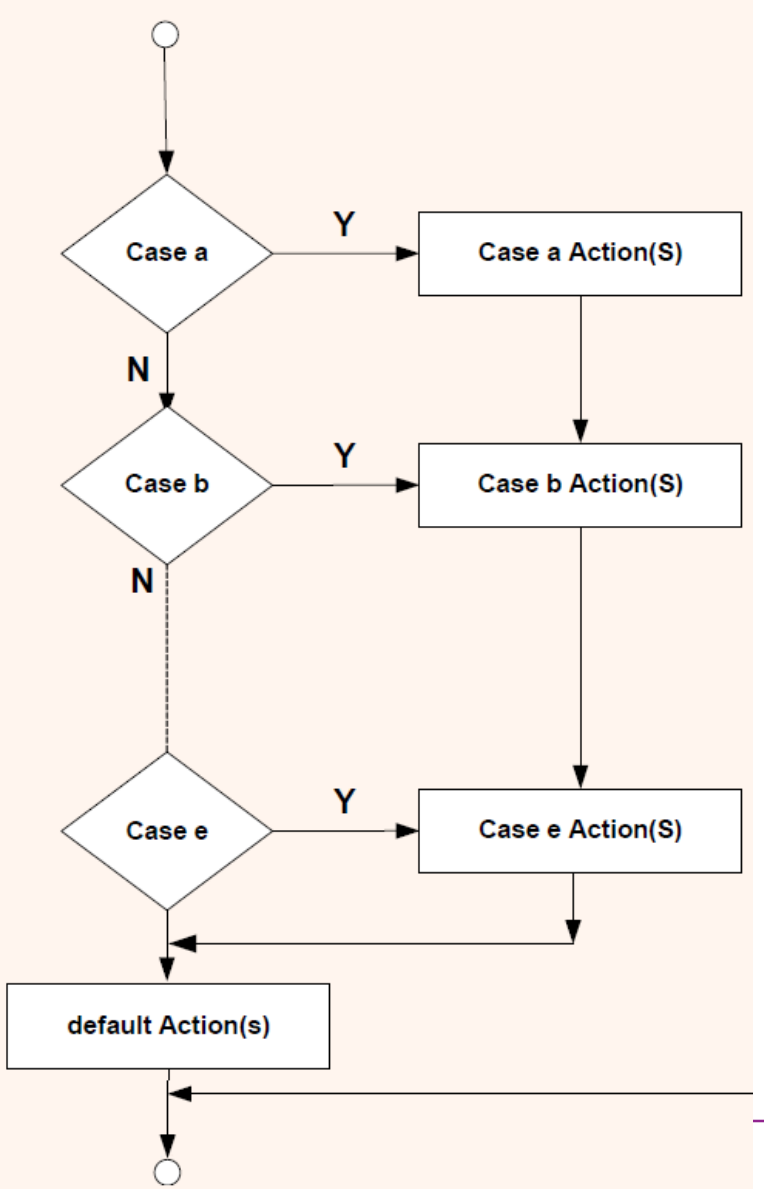
- نکته:
- امکان استفاده از انواع اعشاری به عنوان پارامتر switch وجود ندارد
- در مثال فوق، i نمی‌تواند از نوع float یا double باشد

مثال: ساخت منوی کنسولی

```
int num1, num2;
cout << "Please enter two integers: \n";
cin >> num1 >> num2;
cout << "Please choose an operation: \n";
cout << "1) ADD" << '\n';
cout << "2) SUBTRACT" << '\n';
cout << "3) MULTIPLY" << '\n';
cout << "4) DIVISION" << '\n';
int result = 0;
int menu;
cin >> menu;
switch (menu) {
case 1: result = num1 + num2; break;
case 2: result = num1 - num2; break;
case 3: result = num1 * num2; break;
case 4: result = num1 / num2; break;
default: cout << "Invalid menu number \n";
}
cout<<"Result = " << result;
```

نکته: break را در انتهای هر case فراموش نکنید





مثال

```
char grade = 'A';
switch (grade) {
case 'A':
  cout << "Excellent!" << endl;
  break;
case 'B':
case 'C':
  cout << "Well done" << endl;
  break;
case 'D':
  cout << "You passed" << endl;
  break;
case 'F':
  cout << "Better try again" << endl;
  break;
default:
  cout << "Invalid grade" << endl;
}
```

عملگرهای بیتی

عملگرهای بیتی

- عملگرهایی که کار با تک‌تک بیت‌های موجود در یک متغیر را ممکن می‌سازند
  - مثلاً عملگری که دو متغیر از جنس عدد صحیح را «بیت‌به‌بیت» با هم OR کند
- کاربرد: انجام عمل سطح پایین در سطح بیت‌ها
  - مثلاً می‌توانیم ۸ متغیر boolean را در یک بایت نگهداری کنیم
- به مرور زمان، کاربرد عملگرهای بیتی کمتر شده است
  - برنامه‌نویسان کمتر از این عملگرها استفاده می‌کنند (عملیات سطح پایینی هستند)
  - البته همچنان برای بهینه‌سازی برخی عملیات، از این‌گونه عملگرها استفاده می‌شود
  - مثلاً برای برنامه‌نویسی در سخت‌افزارهای بسیار کوچک و محدود که حافظه و منابع کمی دارند

عملگرهای بیتی

Operator	Symbol	Form	Operation
Left shift	<<	x << y	بیت‌های x ، y بیت به چپ منتقل می‌شوند (شیفت)
Right shift	>>	x >> y	بیت‌های x ، y بیت به راست منتقل می‌شوند (شیفت)
Bitwise NOT	~	~x	همه بیت‌های x نقیض می‌شوند
Bitwise AND	&	x & y	بیت‌های x و y ، به صورت منطقی AND می‌شوند
Bitwise OR	|	x | y	بیت‌های x و y ، به صورت منطقی OR می‌شوند
Bitwise XOR	^	x ^ y	بیت‌های x و y ، به صورت منطقی XOR می‌شوند

- XOR وقتی مقدار true برمی‌گرداند که فقط و فقط یکی از عملوندهایش true باشد

مثال برای عملگرهای بیتی

```
unsigned int a = 60;    // 60 = 0011 1100
unsigned int b = 13;    // 13 = 0000 1101
int c = 0;

c = a & b;              // 12 = 0000 1100
c = a | b;              // 61 = 0011 1101
c = a ^ b;              // 49 = 0011 0001
c = ~a;                 // -61 = 1100 0011
c = a << 2;             // 240 = 1111 0000
c = a >> 2;             // 15 = 0000 1111
```

- امکان استفاده از عملگرهای بیتی برای اعداد اعشاری وجود ندارد
  (این کار به خطای کامپایل منجر می‌شود)
  - چرا؟

مرور اولویت عملگرها

Operator Type	Operators	Precedence
Parentheses	( )	Highest Precedence
Unary	! , ++ , -- , -	
Arithmetic Multiplicative Additive	 * , / , % + , -	
Relational Inequality Equality	 < , > , <= , >= == , !=	
Logical And Or	 && ||	
Conditional	?:	
Assignment	= , += , -= , *= , /= , %=	Lowest Precedence

جمع‌بندی

جمع‌بندی

- دستورهای cin و cout
- ساختارهای شرطی
  - ساختار if else
  - دستور switch
- ساختارهای شرطی تودرتو
- دستورات ترکیبی (blocks)
- عملگرهای مقایسه‌ای
- عملگرهای منطقی
- عملگرهای بیتی

- C How to Program (Deitel & Deitel), 7th edition

Chapter 3

جستجوی بیشتر

- تعیین قالب خروجی برای دستور **cout** چگونه است؟ (formatted output)
  - می‌دانیم در printf این کار با کمک پارامتر اول رشته‌ای مشخص می‌شد (مثلاً %5.2f)
  - مثال درباره cout :
    - cout << setprecision(3) << 2.71828;
    - cout<<setw(10)<<"ten"<<"four"<<"four";
    - cout << false<<endl<<std::boolalpha <<false<<endl;
- با عملگرهای مختلف آشنا شدیم. معنای Operator Overloading چیست؟
  - مثلاً چه ارتباطی بین >> در دستور cout و عملگر شیفت بیتی وجود دارد؟

پایان