Programming and Data Structures - I

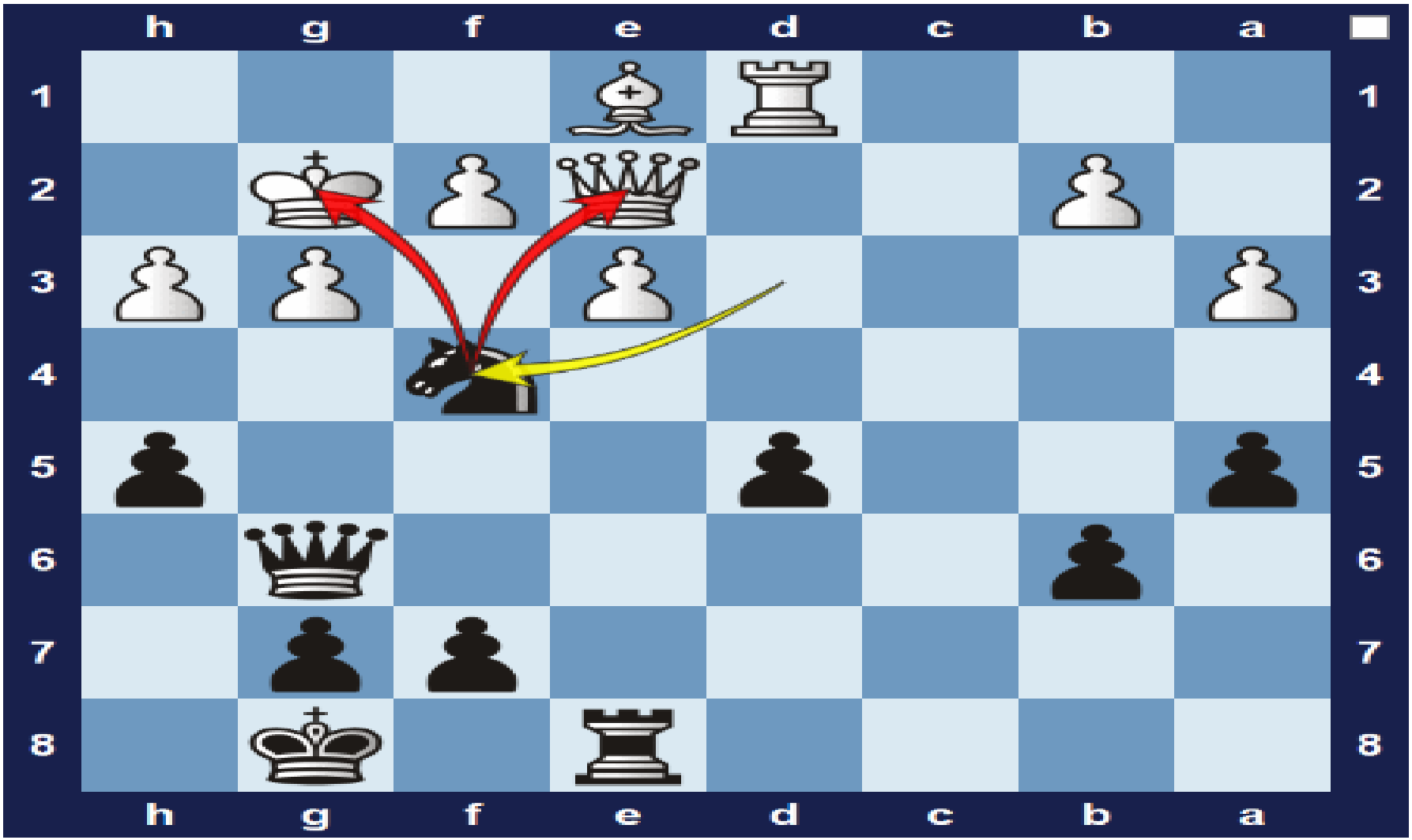
Lecture 8

Kripabandhu Ghosh

CDS, IISER Kolkata

DECISION MAKING AND BRANCHING

Fork[1]



[1] https://chessfox.com/fork/

Fork[2]

[2] https://www.youtube.com/c/ChessTalk

Theory and Coding

Simple *if* statement

Format

```
if(test expression)
{
   true-block;
}
statement-out;
```

- **true-block**: contains one or more statements; executed only when *test expression* is *true*
- **statement-out**:
  - Control goes to this after if statement is executed
  - If *test expression* is *true*, *true-block* and *statement-out* are executed
  - If *test expression* is *false*, *true-block* is not executed but *statement-out* is executed

Drawback

No statements which will be executed only if *test expression* is *false*

Simple *if* statement

Program

```
#include<stdio.h>

int main()
{
   int a = 5;
   if(a > 0)
   {
      printf("True!\n");
   }
   printf("Will be printed anyway!\n");
   return 0;
}
```

Output

```
True!
Will be printed anyway!
```

if..else statement

Format

```
if(test expression)
{
   true-block;
}
else
{
   false-block;
}
statement-out;
```

- **true-block**: contains one or more statements; executed only when *test expression* is *true*
- **false-block**: contains one or more statements; executed only when *test expression* is *false*
- **statement-out**:
  - Control goes to this after if statement is executed
  - If *test expression* is *true*, *true-block* and *statement-out* are executed
  - If *test expression* is *false*, *false-block* and *statement-out* are executed

if..else statement

Program

```
#include<stdio.h>

int main()
{
   int a = 5;
   if(a > 6)
   {
      printf("True!\n");
   }
   else
   {
      printf("False!\n");
   }
   printf("Will be printed anyway!\n");
   return 0;
}
```

Output

```
False!
Will be printed anyway!
```

Multiple conditions

Program

```
#include<stdio.h>

int main()
{
   int a = 5;
   if(a > 0 && a < 10)
   {
      printf("True!\n");
   }
   //Alternative
   if(a > 0)
   {
      if( a < 10)
      {
         printf("True!\n");
      }
   }
   return 0;
}
```

Output

```
True!
True!
```

No brace anomaly

Program

```
#include<stdio.h>

int main()
{
   int a = 3;
   if(a > 0)
      printf("True!\n");
   else
      printf("False!\n");
      printf("I am in else!\n");
   return 0;
}
```

No brace anomaly

Program

```
#include<stdio.h>

int main()
{
   int a = 3;
   if(a > 0)
      printf("True!\n");
   else
      printf("False!\n");
      printf("I am in else!\n");
   return 0;
}
```

Output

```
True!
I am in else!
```

No brace anomaly (contd.)

Program

```
#include<stdio.h>

int main()
{
   int gender = 1, age = 58; //1 = female, 0 = male
   float concession = 0.0;

   if(gender == 0)
      if(age > 58)
         concession = 0.3;
   else
      concession = 0.2;

   printf("Concession = %f\n", concession);
   return 0;
}
```

No brace anomaly (contd.)

Program

```
#include<stdio.h>

int main()
{
   int gender = 1, age = 58; //1 = female, 0 = male
   float concession = 0.0;

   if(gender == 0)
      if(age > 58)
         concession = 0.3;
   else
      concession = 0.2;

   printf("Concession = %f\n", concession);
   return 0;
}
```

Output

```
Concession = 0.000000
```

No brace anomaly (corrected)

Program

```
#include<stdio.h>

int main()
{
   int gender = 1, age = 58; //1 = female, 0 = male
   float concession = 0.0;

   if(gender == 0){
      if(age > 58)
         concession = 0.3;}
   else
      concession = 0.2;

   printf("Concession = %f\n", concession);
   return 0;
}
```

No brace anomaly (corrected)

Program

```
#include<stdio.h>

int main()
{
   int gender = 1, age = 58; //1 = female, 0 = male
   float concession = 0.0;

   if(gender == 0){
      if(age > 58)
         concession = 0.3;}
   else
      concession = 0.2;

   printf("Concession = %f\n", concession);
   return 0;
}
```

Output

```
Concession = 0.200000
```

No brace: another example

Program

```
#include<stdio.h>

void main()
{
   int colour = 2; //BLUE = 1, GREEN = 4, RED = 2, ORANGE = 3
   if(colour != 3)
      if(colour != 2)
         if(colour != 4)
            printf("BLUE\n");
         else
            printf("GREEN\n");
      else
         printf("RED\n");
   else
      printf("ORANGE\n");
}
```

Output

```
RED
```

No Braces: important points

Attention!

- Only the statement immediately following an if/else is within the scope
- The closest if-else pair together
- Indentation is not relevant

No Braces: important points

Attention!

- Only the statement immediately following an if/else is within the scope
- The closest if-else pair together
- Indentation is not relevant

Better to use braces, if unsure!

If..else..if

Format

```
if(condition 1)
   block 1;
else if (condition 2)
      block 2;
   else if (condition 3)
         block 3;
         ...
      else if (condition n)
            block n;
            else
               default block;
```

- *block 1* is executed if *condition 1* is True
- *block 2* is executed if *condition 1* is False and *condition 2* is True, and so on
- In general, *block i* is executed if *condition i* is True and conditions 1, 2, ..., $i-1$ are False
- *default block* is executed if all the conditions 1, 2, ..., n are False

If..else..if (example)

Program

```
#include<stdio.h>

void main()
{
   char grade;
   int marks = 5;

   if(marks > 90)
      grade = 'S';
   else if(marks > 80)
         grade = 'A';
      else if(marks > 70)
            grade = 'B';
         else if(marks > 60)
               grade = 'C';
            else if(marks > 50)
                  grade = 'D';
               else
                     grade = 'F';

   printf("Grade = %c\n", grade);
}
```

Output

```
Grade = F
```

If..else..if (example with block)

Program

```
#include<stdio.h>

void main()
{
  char grade;
  int marks = 95;

  if(marks > 90) {
    grade = 'S';
    printf("The highest Grade!\n");}
  else if(marks > 80)
      grade = 'A';
    else if(marks > 70)
        grade = 'B';
      else if(marks > 60)
          grade = 'C';
        else if(marks > 50)
            grade = 'D';
          else{
              grade = 'F';
              printf("Failed!!\n");}

  printf("Grade = %c\n", grade);
}
```

Output

```
The highest Grade!
Grade = S
```

switch statement

Format

```
switch (expression)
{
   case value 1: block 1;
                 break;
   case value 2: block 2;
                 break;
      ...
   default: default block;
                 break;
}
```

- **expression**: integer expression or character
- value 1, value 2 etc.: integer
- **default**: considered if none of the *'value i'*s satisfy *expression* value
- block 1, block 2 etc.: may contain zero or more statements
- default block: executed for *default*

switch statement: examples

Program

```
#include<stdio.h>
#include<math.h>

int main()
{
  char grade;
  int marks = 83;
  int gradation = ceil(marks/10.0);

  switch(gradation)
  {
    case 10: grade = 'S';
             break;
    case 9: grade = 'A';
             break;
    case 8: grade = 'B';
             break;
    case 7: grade = 'C';
             break;
    case 6: grade = 'D';
             break;
    default: grade = 'F';
             break;
  }
  printf("Grade = %c\n", grade);
  return 0;
}
```

Output

```
Grade = A
```

switch statement: examples

Program

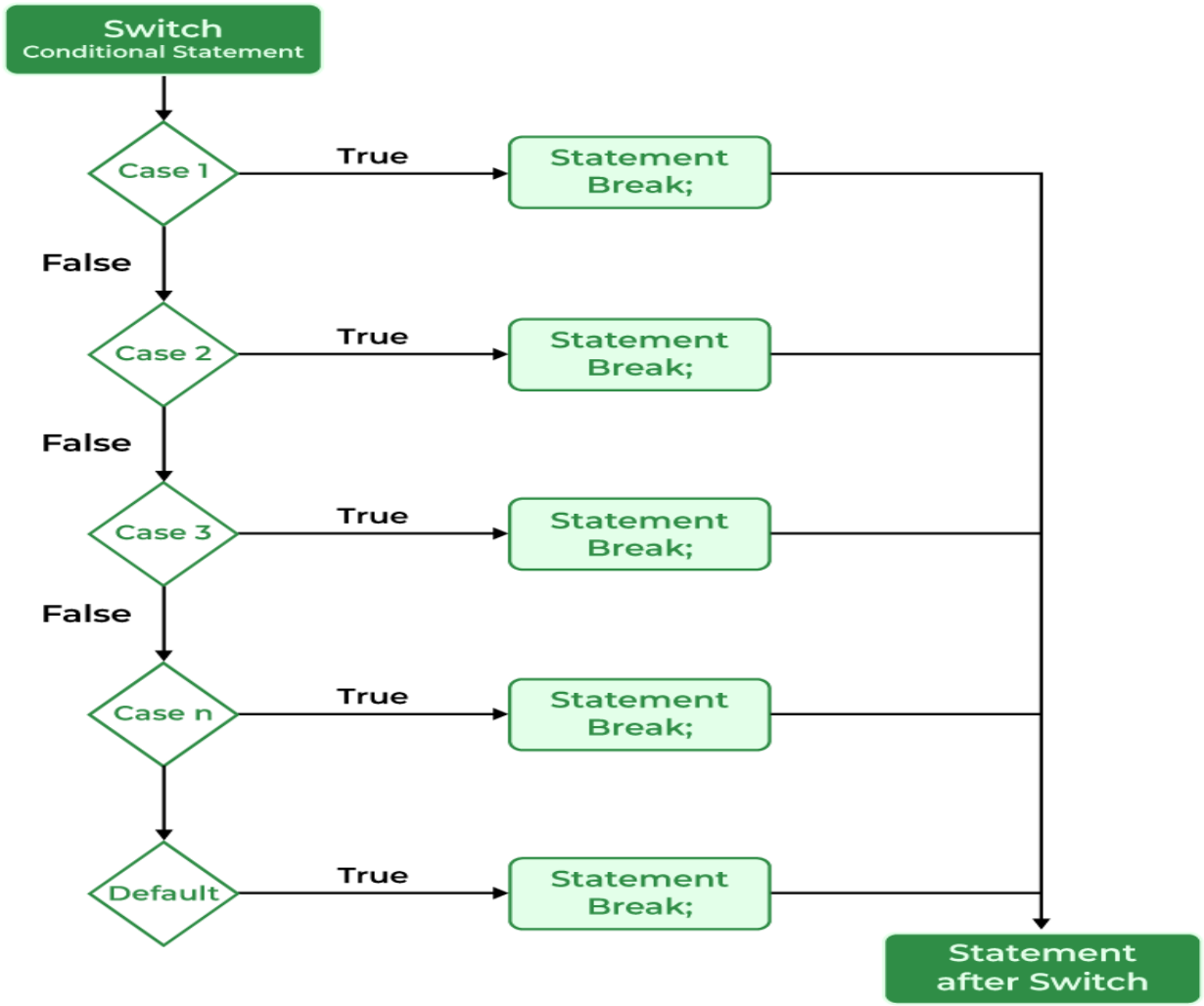
```
#include<stdio.h>
#include<math.h>

int main()
{
  char choice = 'A';
  switch(choice)
  {
    case 'A': printf("A for Android!\n");
              break;
    case 'B': printf("B for Big Data!\n");
              break;
    case 'C': printf("C for Cloud Computing!\n");
              break;
    case 'S': printf("S for Selfie!\n");
              break;
    default: printf("No idea!!\n");
              break;
}
  return 0;
}
```

Output

```
A for Android!
```

switch statement: schematic[3]



[3] https://www.geeksforgeeks.org/c-switch-statement/

switch statement: features

Advantage

Simplifies complex branching

Drawback

The choice and the condition values need to be integers

? operator

Format

(conditional expression)?expression1 : expression2

- If *conditional expression* is True, *expression1* is evaluated
- If *conditional expression* is False, *expression2* is evaluated

? operator: example

Program

```
#include<stdio.h>
#include<math.h>

int main()
{
   int a = 5, b = 7, max;
   if(a > b)
   {
      max = a;
   }
   else
   {
      max = b;
   }
   printf("Max = %d\n", max);
   //Alternative
   max = (a > b)? a : b;
   printf("Max = %d\n", max);
   return 0;
}
```

Output

```
Max = 7
Max = 7
```

Multivalued functions

$$f(x) = \begin{cases} x & \text{if } x < 0 \\ x^2 & \text{if } x = 0 \\ 3x^3 & \text{if } x > 0 \end{cases}$$

? operator: example

Program

```
#include<stdio.h>
#include<math.h>

int main()
{
  int x = 5, f;
  if(x < 0)
  {
    f = x;
  }
  else
  {
    if(x == 0)
    {
      f = x*x;
    }
    else
    {
      f = 3*x*x*x;
    }
  }
  printf("f = %d\n", f);
  //Alternative
  f = (x == 0)? (x*x) : ((x < 0)? (x) : (3*x*x*x));
  printf("f = %d\n", f);
  return 0;
}
```

Output

```
f = 375
f = 375
```

? operator: features

Advantage

Concise representation

Drawback

For multivalues expressions, readability can be an issue

goto statement

Unconditional branching

goto statement: example

Program

```
#include<stdio.h>

int main()
{
   goto print;
   printf("Why use GOTO?\n");
   print: printf("Well I am using GOTO!\n");
   return 0;
}
```

Output

```
Well I am using GOTO!
```

Disclaimer[4]

- Formally, the *goto* statement is never necessary, and in practice it is almost always easy to write code without it.
- Although we are not dogmatic about the matter, it does seem that *goto* statements should be used rarely, if at all.

[4] The C Programming Language. Kernighan and Ritchie.

C you!!