

# Still Looping with Mr C

ESC101: Fundamentals of Computing

Purushottam Kar

# Announcements - Holiday

- Extra lecture on Saturday 08 September, 2018
  - 12 noon, L20 (same as usual)
  - Scheduled by DoAA, not by me – I like to sleep on Sat too ☹
- Extra lab for B1, B2, B3 on Saturday 08 September
  - 2PM – 5PM, New Core Labs CC-02 (same as usual)



# Announcements - AT



# Announcements - AT

- AT offers made last evening



# Announcements - AT

- AT offers made last evening
- Deadline for accepting offers - Thu, 06 Sep 10PM IST



# Lab Exam (Sun, 09 Sep)

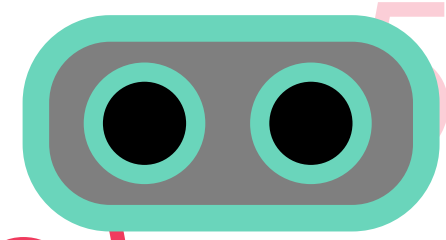
- Morning exam (Wed, Thu batches)
  - 10:30 AM - 1:30 PM – starts 10:30 AM sharp
  - **CC-01**: B9, B14 even roll numbers
  - **CC-02**: B7, B10, B11
  - **CC-03**: B12
  - **MATH-LINUX**: B8, B14 odd roll numbers
- Go see your room during this week's lab
- Be there 15 minutes before your exam 10:15AM
- Cannot switch to afternoon session



# Lab Exam (Sun, 09 Sep)

- Morning exam (Wed, Thu batches)
  - 10:30 AM - 1:30 PM – starts 10:30 AM sharp
  - **CC-01**: B9, {B14 even roll numbers}
  - **CC-02**: B7, B10, B11
  - **CC-03**: B12
  - **MATH-LINUX**: B8, {B14 odd roll numbers}
- Go see your room during this week's lab
- Be there 15 minutes before your exam 10:15AM
- Cannot switch to afternoon session



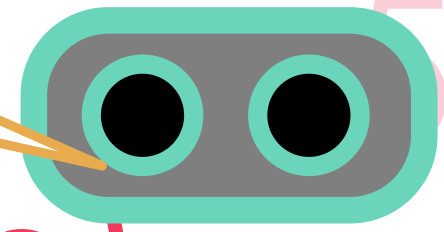


# Lab Exam (Sun, 09 Sep)

- Morning exam (Wed, Thu batches)
  - 10:30 AM - 1:30 PM – starts 10:30 AM sharp
  - **CC-01**: B9, {B14 even roll numbers}
  - **CC-02**: B7, B10, B11
  - **CC-03**: B12
  - **MATH-LINUX**: B8, {B14 odd roll numbers}
- Go see your room during this week's lab
- Be there 15 minutes before your exam 10:15AM
- Cannot switch to afternoon session



Brackets matter 😊



# Lab Exam (Sun, 09 Sep)

- Morning exam (Wed, Thu batches)
  - 10:30 AM - 1:30 PM – starts 10:30 AM sharp
  - **CC-01**: B9, {B14 even roll numbers}
  - **CC-02**: B7, B10, B11
  - **CC-03**: B12
  - **MATH-LINUX**: B8, {B14 odd roll numbers}
- Go see your room during this week's lab
- Be there 15 minutes before your exam 10:15AM
- Cannot switch to afternoon session



# Lab Exam (Sun, 09 Sep)

- Afternoon exam (Mon, Tue batches)
  - 2 PM - 5 PM – starts 2 PM sharp
  - **CC-01**: B1, {B2 even roll numbers}
  - **CC-02**: B4, B5, B6
  - **CC-03**: B3
  - **MATH-LINUX**: B13, {B2 odd roll numbers}
- Go see your room during this week's lab
- Be there 15 minutes before your exam 1:45 PM
- Cannot switch to morning session



# Lab Exam (Sun, 09 Sep)



# Lab Exam (Sun, 09 Sep)

- Syllabus – till loops (no arrays)



# Lab Exam (Sun, 09 Sep)

- Syllabus – till loops (no arrays)
- Open handwritten notes – However, **NO** printouts, photocopies, slides, websites, mobile phone, Ipad



# Lab Exam (Sun, 09 Sep)

- Syllabus – till loops (no arrays)
- Open handwritten notes – However, **NO** printouts, photocopies, slides, websites, mobile phone, Ipad
- **USE OF ANY OF ABOVE WILL BE CONSIDERED CHEATING**



# Lab Exam (Sun, 09 Sep)

- Syllabus – till loops (no arrays)
- Open handwritten notes – However, **NO** printouts, photocopies, slides, websites, mobile phone, Ipad
- **USE OF ANY OF ABOVE WILL BE CONSIDERED CHEATING**
- Prutor CodeBook will be unavailable during lab exam



# Lab Exam (Sun, 09 Sep)

- Syllabus – till loops (no arrays)
- Open handwritten notes – However, **NO** printouts, photocopies, slides, websites, mobile phone, Ipad
- **USE OF ANY OF ABOVE WILL BE CONSIDERED CHEATING**
- Prutor CodeBook will be unavailable during lab exam
- Exam will be like labs - marks for passing test cases



# Lab Exam (Sun, 09 Sep)

- Syllabus – till loops (no arrays)
- Open handwritten notes – However, **NO** printouts, photocopies, slides, websites, mobile phone, Ipad
- **USE OF ANY OF ABOVE WILL BE CONSIDERED CHEATING**
- Prutor CodeBook will be unavailable during lab exam
- Exam will be like labs - marks for passing test cases
- Marks for writing clean indented code, proper variable names, a few comments – illegible code poor marks



# Break and Continue

8



# Break and Continue

8

Break helps us exit loop immediately



# Break and Continue

8

Break helps us exit loop immediately

In for loops, even update\_expr or stop\_expr not checked – just exit



# Break and Continue

8

Break helps us exit loop immediately

In for loops, even update\_expr or stop\_expr not checked – just exit

In while, do-while loops, even stop\_expr not checked – just exit



# Break and Continue

8

Break helps us exit loop immediately

In for loops, even update\_expr or stop\_expr not checked – just exit

In while, do-while loops, even stop\_expr not checked – just exit

Continue helps us skip the rest of the body of loop



# Break and Continue

8

Break helps us exit loop immediately

In for loops, even update\_expr or stop\_expr not checked – just exit

In while, do-while loops, even stop\_expr not checked – just exit

Continue helps us skip the rest of the body of loop

In for loops, after Mr C receives a continue statement, he evaluates the update\_expr, then checks the stop\_expr and so on ...



# Break and Continue

8

Break helps us exit loop immediately

In for loops, even update\_expr or stop\_expr not checked – just exit

In while, do-while loops, even stop\_expr not checked – just exit

Continue helps us skip the rest of the body of loop

In for loops, after Mr C receives a continue statement, he evaluates the update\_expr, then checks the stop\_expr and so on ...

In while loops, after Mr C receives a continue statement, he checks the stop\_expr



# Break and Continue

8

Break helps us exit loop immediately

In for loops, even update\_expr or stop\_expr not checked – just exit

In while, do-while loops, even stop\_expr not checked – just exit

Continue helps us skip the rest of the body of loop

In for loops, after Mr C receives a continue statement, he evaluates the update\_expr, then checks the stop\_expr and so on ...

In while loops, after Mr C receives a continue statement, he checks the stop\_expr

Loop not exited just because of continue, stop\_expr still controls exit



# Break and Continue

8

Break helps us exit loop immediately

In for loops, even update\_expr or stop\_expr not checked – just exit

In while, do-while loops, even stop\_expr not checked – just exit

Continue helps us skip the rest of the body of loop

In for loops, after Mr C receives a continue statement, he evaluates the update\_expr, then checks the stop\_expr and so on ...

In while loops, after Mr C receives a continue statement, he checks the stop\_expr

Loop not exited just because of continue, stop\_expr still controls exit

**Warning:** Break legal only in body of loops and switch



# Break and Continue

8

Break helps us exit loop immediately

In for loops, even update\_expr or stop\_expr not checked – just exit

In while, do-while loops, even stop\_expr not checked – just exit

Continue helps us skip the rest of the body of loop

In for loops, after Mr C receives a continue statement, he evaluates the update\_expr, then checks the stop\_expr and so on ...

In while loops, after Mr C receives a continue statement, he checks the stop\_expr

Loop not exited just because of continue, stop\_expr still controls exit

**Warning:** Break legal only in body of loops and switch

Illegal inside body of if, if-else statements



# Break and Continue

8

Break helps us exit loop immediately

In for loops, even update\_expr or stop\_expr not checked – just exit

In while, do-while loops, even stop\_expr not checked – just exit

Continue helps us skip the rest of the body of loop

In for loops, after Mr C receives a continue statement, he evaluates the update\_expr, then checks the stop\_expr and so on ...

In while loops, after Mr C receives a continue statement, he checks the stop\_expr

Loop not exited just because of continue, stop\_expr still controls exit

**Warning:** Break legal only in body of loops and switch

Illegal inside body of if, if-else statements

**Warning:** Continue legal only in body of loops



# Break and Continue

8

Break helps us exit loop immediately

In for loops, even update\_expr or stop\_expr not checked – just exit

In while, do-while loops, even stop\_expr not checked – just exit

Continue helps us skip the rest of the body of loop

In for loops, after Mr C receives a continue statement, he evaluates the update\_expr, then checks the stop\_expr and so on ...

In while loops, after Mr C receives a continue statement, he checks the stop\_expr

Loop not exited just because of continue, stop\_expr still controls exit

**Warning:** Break legal only in body of loops and switch

Illegal inside body of if, if-else statements

**Warning:** Continue legal only in body of loops

Illegal inside body of if, if-else, switch statements



# How to avoid Breaking

9



# How to avoid Breaking

9

Take a number, if it is even print “Even”, if it is also divisible by 5, print “Divisible by 10” as well, on a different line



# How to avoid Breaking

9

Take a number, if it is even print “Even”, if it is also divisible by 5, print “Divisible by 10” as well, on a different line

Several ways to solve this problem – one wrong way



# How to avoid Breaking

9

Take a number, if it is even print “Even”, if it is also divisible by 5, print “Divisible by 10” as well, on a different line

Several ways to solve this problem – one wrong way

```
int num = 20;
if(num %2 == 0){
    printf("Even");
    if(num % 5 != 0) break;
    printf("\nDivisible by 10");
}
```



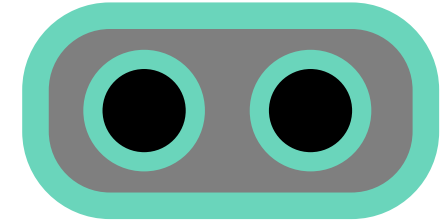
# How to avoid Breaking

9

Take a number, if it is even print “Even”, if it is also divisible by 5, print “Divisible by 10” as well, on a different line

Several ways to solve this problem – one wrong way

```
int num = 20;
if(num %2 == 0){
    printf("Even");
    if(num % 5 != 0) break;
    printf("\nDivisible by 10");
}
```



# How to avoid Breaking

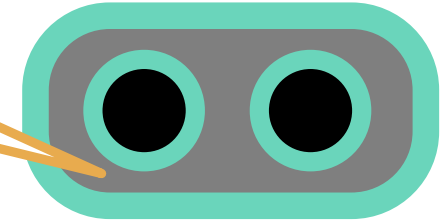
9

Take a number, if it is even print “Even”, if it is also divisible by 5, print “Divisible by 10” as well, on a different line

Several ways to solve this problem – one wrong way

```
int num = 20;
if(num %2 == 0){
    printf("Even");
    if(num % 5 != 0) break;
    printf("\nDivisible by 10");
}
```

Break does not make sense to me inside if-else



# How to avoid Breaking

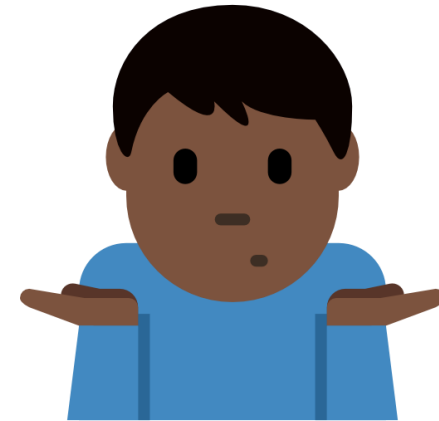
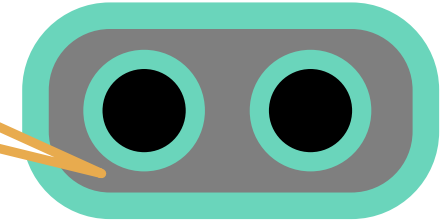
9

Take a number, if it is even print “Even”, if it is also divisible by 5, print “Divisible by 10” as well, on a different line

Several ways to solve this problem – one wrong way

```
int num = 20;
if(num %2 == 0){
    printf("Even");
    if(num % 5 != 0) break;
    printf("\nDivisible by 10");
}
```

Break does not make sense to me inside if-else



# How to avoid Breaking

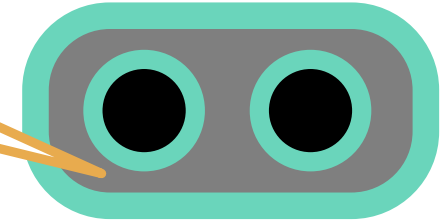
9

Take a number, if it is even print “Even”, if it is also divisible by 5, print “Divisible by 10” as well, on a different line

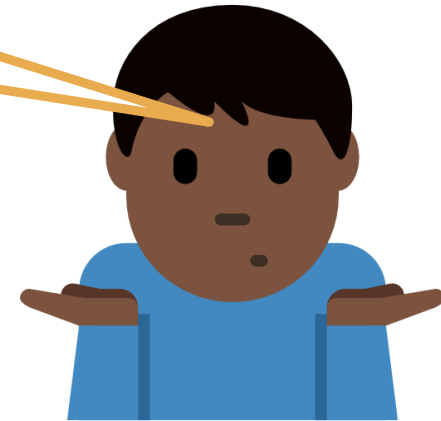
Several ways to solve this problem – one wrong way

```
int num = 20;
if(num %2 == 0){
    printf("Even");
    if(num % 5 != 0) break;
    printf("\nDivisible by 10");
}
```

Break does not make sense to me inside if-else



What if I want to skip the rest of the body of the if?



# How to avoid Breaking

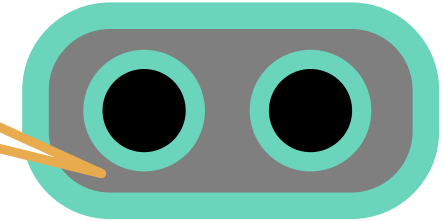
9

Take a number, if it is even print “Even”, if it is also divisible by 5, print “Divisible by 10” as well, on a different line

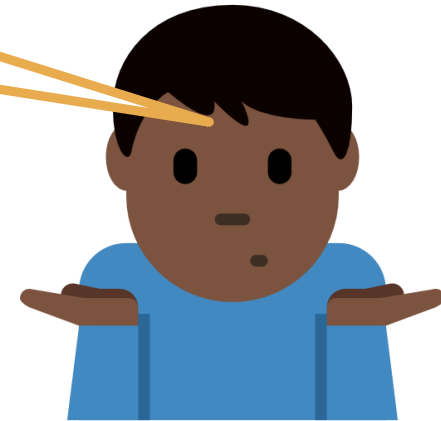
Several ways to solve this problem – one wrong way

```
int num = 20;
if(num %2 == 0){
    printf("Even");
    if(num % 5 != 0) break;
    printf("\nDivisible by 10");
}
```

Break does not make sense to me inside if-else



What if I want to skip the rest of the body of the if?



# How to avoid Breaking

9

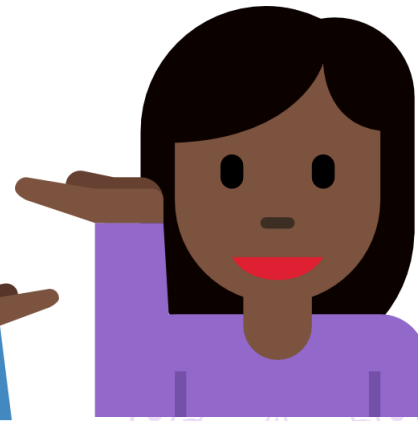
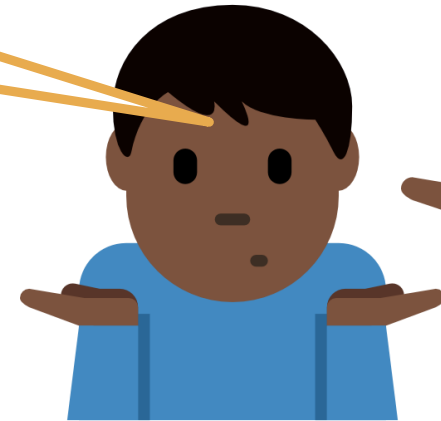
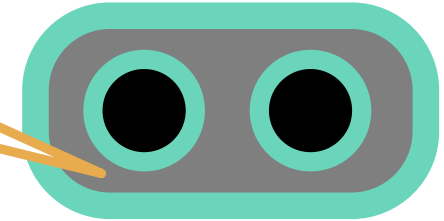
Take a number, if it is even print “Even”, if it is also divisible by 5, print “Divisible by 10” as well, on a different line

Several ways to solve this problem – one wrong way

```
int num = 20;
if(num %2 == 0){
    printf("Even");
    if(num % 5 != 0) break;
    printf("\nDivisible by 10");
}
```

Break does not make sense to me inside if-else

What if I want to skip the rest of the body of the if?



# How to avoid Breaking

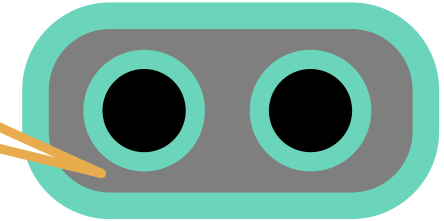
9

Take a number, if it is even print “Even”, if it is also divisible by 5, print “Divisible by 10” as well, on a different line

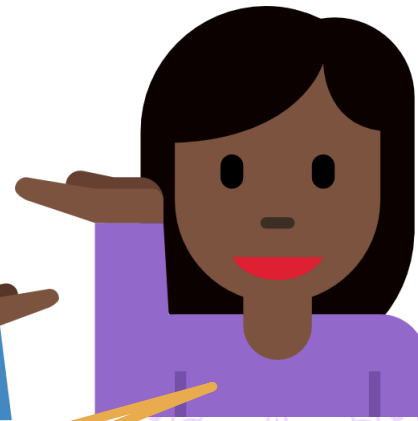
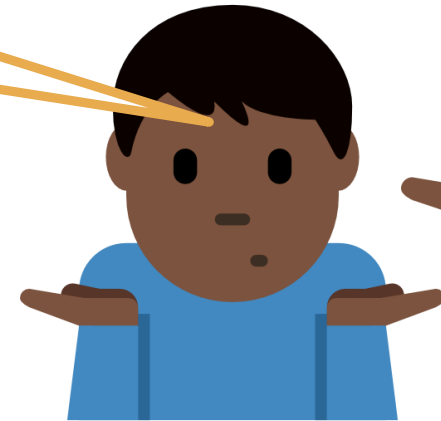
Several ways to solve this problem – one wrong way

```
int num = 20;
if(num %2 == 0){
    printf("Even");
    if(num % 5 != 0) break;
    printf("\nDivisible by 10");
}
```

Break does not make sense to me inside if-else



What if I want to skip the rest of the body of the if?



Use flags



# How to avoid Breaking

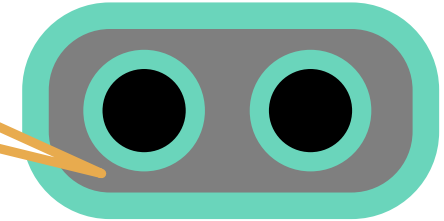
9

Take a number, if it is even print “Even”, if it is also divisible by 5, print “Divisible by 10” as well, on a different line

Several ways to solve this problem – one wrong way

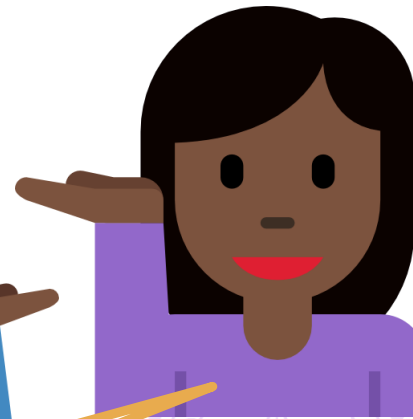
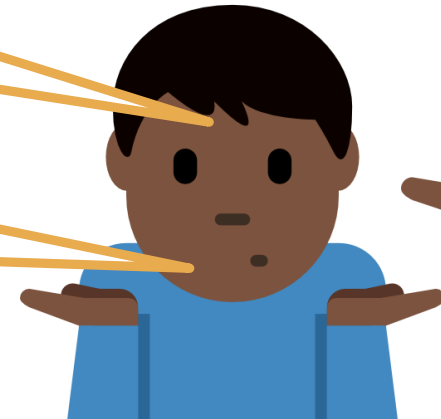
```
int num = 20;
if(num %2 == 0){
    printf("Even");
    if(num % 5 != 0) break;
    printf("\nDivisible by 10");
}
```

Break does not make sense to me inside if-else



What if I want to skip the rest of the body of the if?

What are flags?



Use flags



# Flags

10



# Flags

10

Flags **NOT A KEYWORD** – they are a programming style



# Flags

10

Flags **NOT A KEYWORD** – they are a programming style

As name suggests, they signal important happenings



# Flags

10

Flags **NOT A KEYWORD** – they are a programming style

As name suggests, they signal important happenings

Can be used to avoid using break and continue



# Flags

10

Flags **NOT A KEYWORD** – they are a programming style

As name suggests, they signal important happenings

Can be used to avoid using break and continue

Flags can be integer, long variables – usually 0/1



# Flags

10

Flags **NOT A KEYWORD** – they are a programming style

As name suggests, they signal important happenings

Can be used to avoid using break and continue

Flags can be integer, long variables – usually 0/1

You can give your flag any legal, sensible name you want



# Flags

10

Flags **NOT A KEYWORD** – they are a programming style

As name suggests, they signal important happenings

Can be used to avoid using break and continue

Flags can be integer, long variables – usually 0/1

You can give your flag any legal, sensible name you want

```
int num = 20;
int flag = 0; // Assume not div by 5
if(num %2 == 0){
    printf("Even");
    if(num % 5 == 0) flag = 1;
    if(flag)
        printf("\nDivisible by 10");
}
```

# Flags

10

**CRITICAL:** Always initialize your flags

Flags **N** are a programming style – they

As name suggests, they signal important happenings

Can be used to avoid using break and continue

Flags can be integer, long variables – usually 0/1

You can give your flag any legal, sensible name you want

```
int num = 20;
int flag = 0; // Assume not div by 5
if(num %2 == 0){
    printf("Even");
    if(num % 5 == 0) flag = 1;
    if(flag)
        printf("\nDivisible by 10");
}
```

# Flags

10

**CRITICAL:** Always initialize your flags – they

Flags **N** are a programming style

As num is not multiple of 5, flag may contain garbage value

Can be used to avoid using break and continue

Flags can be integer, long variables – usually 0/1

You can give your flag any legal, sensible name you want

```
int num = 20;
int flag = 0; // Assume not div by 5
if(num %2 == 0){
    printf("Even");
    if(num % 5 == 0) flag = 1;
    if(flag)
        printf("\nDivisible by 10");
}
```

# Flags

10

Lovingly called *default value* of the flag

**CRITICAL:** Always initialize your flags

Flags **N** initialize your flags – they are a programming style

As If you don't initialize and num is not multiple of 5, flag may contain garbage value

Can be used to avoid using break and continue

Flags can be integer, long variables – usually 0/1

You can give your flag any legal, sensible name you want

```
int num = 20;
int flag = 0; // Assume not div by 5
if(num % 2 == 0){
    printf("Even");
    if(num % 5 == 0) flag = 1;
    if(flag)
        printf("\nDivisible by 10");
}
```

# Flags

10

Lovingly called *default value* of the flag

**CRITICAL:** Always initialize your flags

Flags **N** initialize your flags – they are a programming style

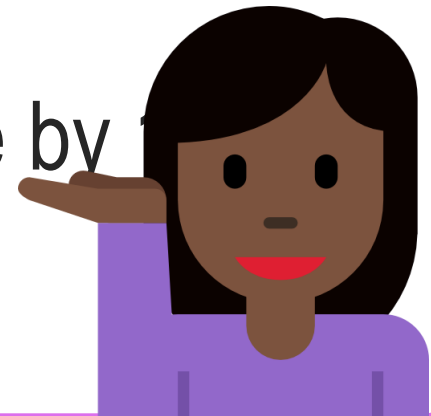
As num is not multiple of 5, flag may contain garbage value

Can be used to avoid using break and continue

Flags can be integer, long variables – usually 0/1

You can give your flag any legal, sensible name you want

```
int num = 20;
int flag = 0; // Assume not div by 5
if(num % 2 == 0){
    printf("Even");
    if(num % 5 == 0) flag = 1;
    if(flag)
        printf("\nDivisible by 10");
}
```



# Flags

10

Lovingly called *default value* of the flag

**CRITICAL:** Always initialize your flags – they

Flags **N** are a programming style

As num is not multiple of 5, flag may contain garbage value

Can be used to avoid using break and continue

Flags can be integer, long variables – usually 0/1

You can give your flag any legal, sensible name you want

```
int num = 20;
int flag = 0; // Assume not div by 5
if(num % 2 == 0){
    printf("Even");
    if(num % 5 == 0) flag = 1;
```

Could have also named this flag isDivBy5 – more descriptive name

```
printf("divisible by 5");
}
```



# Avoiding Continue using Flags

11



# Avoiding Continue using Flags

11

Read 100 integers and print sum of only positive numbers



# Avoiding Continue using Flags

11

Read 100 integers and print sum of only positive numbers

```
int sum = 0, i, num;
for(i = 1; i <= 100; i++){
    scanf("%d", &num);
    if (num < 0)
        continue;
    sum += num;
}
```



# Avoiding Continue using Flags

11

Read 100 integers and print sum of only positive numbers

```
int sum = 0, i, num;
for(i = 1; i <= 100; i++){
    scanf("%d", &num);
    if (num < 0)
        continue;
    sum += num;
}
```

```
int sum = 0, i, num, flag;
for(i = 1; i <= 100; i++){
    scanf("%d", &num);
    if (num < 0) flag = 0;
    else flag = 1;
    if(flag) sum += num;
}
```



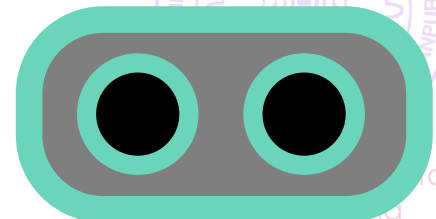
# Avoiding Continue using Flags

11

Read 100 integers and print sum of only positive numbers

```
int sum = 0, i, num;
for(i = 1; i <= 100; i++){
    scanf("%d", &num);
    if (num < 0)
        continue;
    sum += num;
}
```

```
int sum = 0, i, num, flag;
for(i = 1; i <= 100; i++){
    scanf("%d", &num);
    if (num < 0) flag = 0;
    else flag = 1;
    if(flag) sum += num;
}
```



# Avoiding Continue using Flags

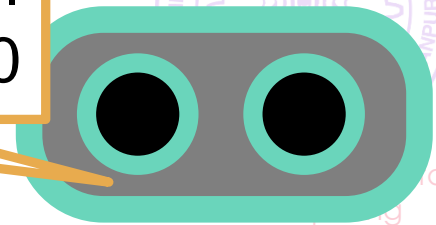
11

Read 100 integers and print sum of only positive numbers

```
int sum = 0, i, num;
for(i = 1; i <= 100; i++){
    scanf("%d", &num);
    if (num < 0)
        continue;
    sum += num;
}
```

```
int sum = 0, i, num, flag;
for(i = 1; i <= 100; i++){
    scanf("%d", &num);
    if (num < 0) flag = 0;
    else flag = 1;
    if(flag) sum += num;
}
```

Warning: need else here since I will not skip statements even if num < 0



# Avoiding Continue using Flags

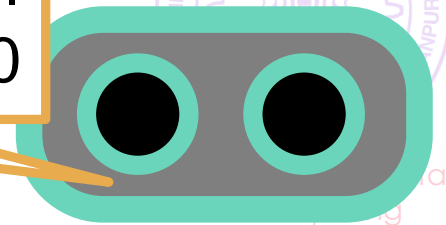
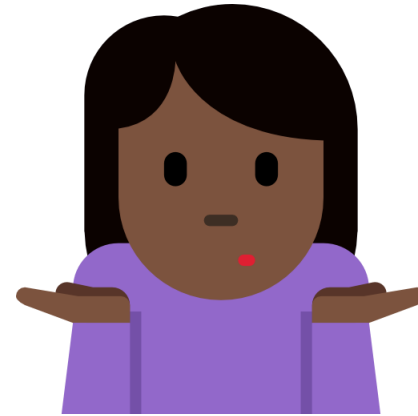
11

Read 100 integers and print sum of only positive numbers

```
int sum = 0, i, num;
for(i = 1; i <= 100; i++){
    scanf("%d", &num);
    if (num < 0)
        continue;
    sum += num;
}
```

```
int sum = 0, i, num, flag;
for(i = 1; i <= 100; i++){
    scanf("%d", &num);
    if (num < 0) flag = 0;
    else flag = 1;
    if(flag) sum += num;
}
```

Warning: need else here since I will not skip statements even if num < 0



# Avoiding Continue using Flags

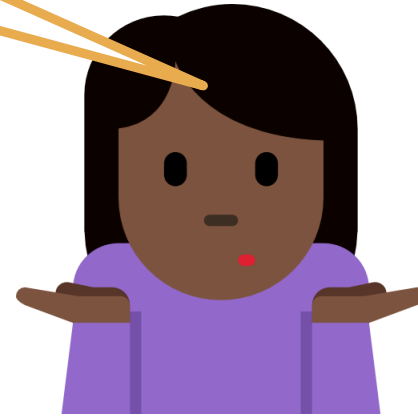
11

Read 100 integers and print sum of only positive numbers

```
int sum = 0, i, num;
for(i = 1; i <= 100; i++){
    scanf("%d", &num);
    if (num < 0)
        continue;
    sum += num;
}
```

```
int sum = 0, i, num, flag;
for(i = 1; i <= 100; i++){
    scanf("%d", &num);
    if (num < 0) flag = 0;
    else flag = 1;
    if(flag) sum += num;
}
```

Why did we not initialize flag here?



Warning: need else here since I will not skip statements even if num < 0



# Avoiding Continue using Flags

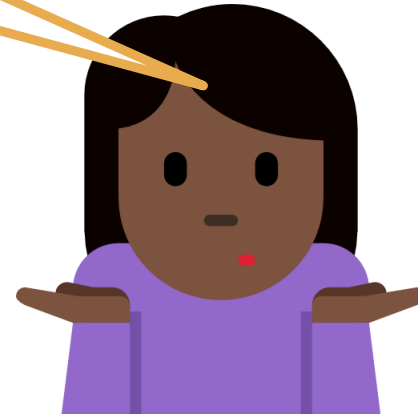
11

Read 100 integers and print sum of only non-negative numbers

```
int sum = 0, i, num;
for(i = 1; i <= 100; i++){
    scanf("%d", &num);
    if (num < 0)
        continue;
    sum += num;
}
```

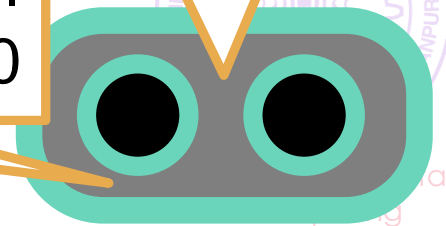
```
int sum = 0, i, num, flag;
for(i = 1; i <= 100; i++){
    scanf("%d", &num);
    if (num < 0) flag = 0;
    else flag = 1;
    if(flag) sum += num;
}
```

Why did we not initialize flag here?



I should have, for safety, but notice, flag always gets set before getting checked

Warning: need else here since I will not skip statements even if num < 0



# Avoiding Continue using Flags

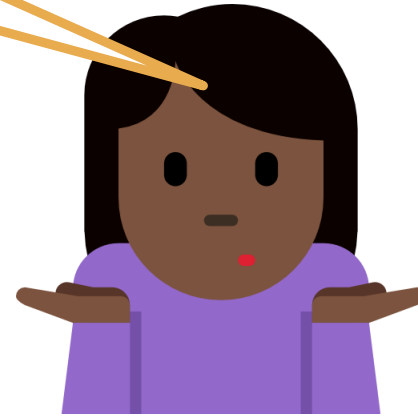
11

Read 100 integers and print sum of only non-negative numbers

```
int sum = 0, i, num;
for(i = 1; i <= 100; i++){
    scanf("%d", &num);
    if (num < 0)
        continue;
    sum += num;
}
```

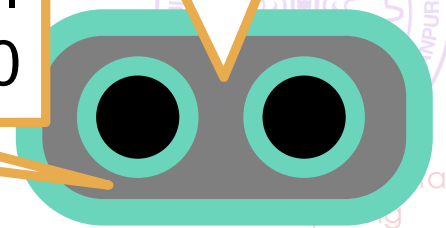
```
int sum = 0, i, num, flag = 0;
for(i = 1; i <= 100; i++){
    scanf("%d", &num);
    if (num < 0) flag = 0;
    else flag = 1;
    if(flag) sum += num;
}
```

Why did we not initialize flag here?



I should have, for safety, but notice, flag always gets set before getting checked

Warning: need else here since I will not skip statements even if num < 0



# Avoiding Break using Flags

12



# Avoiding Break using Flags

12

Read integers till you get -1 and print their sum



# Avoiding Break using Flags

12

Read integers till you get -1 and print their sum

```
int num, sum = 0;
while(1){
    scanf("%d", &num);
    if(num == -1) break;
    sum += num;
}
printf("%d",sum);
```



# Avoiding Break using Flags

12

Read integers till you get -1 and print their sum

```
int num, sum = 0;
while(1){
    scanf("%d", &num);
    if(num == -1) break;
    sum += num;
}
printf("%d",sum);
```

```
int num, sum = 0, flag = 1;
while(flag){
    scanf("%d", &num);
    if(num == -1) flag = 0;
    else sum += num;
}
printf("%d",sum);
```



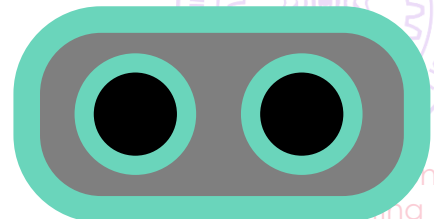
# Avoiding Break using Flags

12

Read integers till you get -1 and print their sum

```
int num, sum = 0;
while(1){
    scanf("%d", &num);
    if(num == -1) break;
    sum += num;
}
printf("%d",sum);
```

```
int num, sum = 0, flag = 1;
while(flag){
    scanf("%d", &num);
    if(num == -1) flag = 0;
    else sum += num;
}
printf("%d",sum);
```



# Avoiding Break using Flags

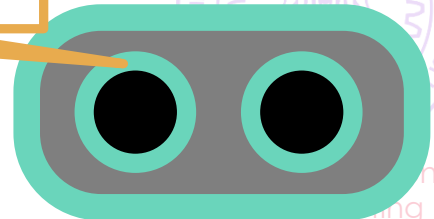
12

Read integers till you get -1 and print their sum

```
int num, sum = 0;
while(1){
    scanf("%d", &num);
    if(num == -1) break;
    sum += num;
}
printf("%d",sum);
```

```
int num, sum = 0, flag = 1;
while(flag){
    scanf("%d", &num);
    if(num == -1) flag = 0;
    else sum += num;
}
printf("%d",sum);
```

Warning: need else here  
since I will not skip these  
statements even if num == -1



# Avoiding Break using Flags

12

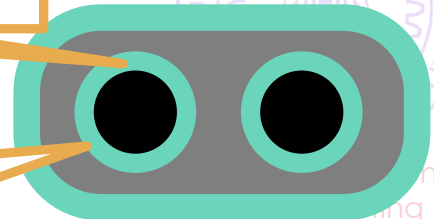
Read integers till you get -1 and print their sum

```
int num, sum = 0;
while(1){
    scanf("%d", &num);
    if(num == -1) break;
    sum += num;
}
printf("%d", sum);
```

```
int num, sum = 0, flag = 1;
while(flag){
    scanf("%d", &num);
    if(num == -1) flag = 0;
    else sum += num;
}
printf("%d", sum);
```

Warning: need else here  
since I will not skip these  
statements even if num == -1

**CRITICAL:** Always  
initialize your flags



# Avoiding Break using Flags

12

Read integers till you get -1 and print their sum

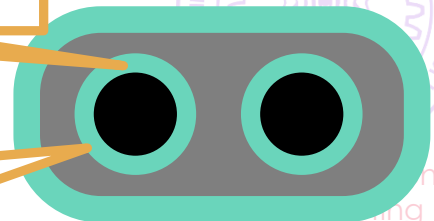
```
int num, sum = 0;
while(1){
    scanf("%d", &num);
    if(num == -1) break;
    sum += num;
```

```
int num, sum = 0, flag = 1;
while(flag){
    scanf("%d", &num);
    if(num == -1) flag = 0;
    else sum += num;
```

To avoid confusion or your yourself forgetting what flag values mean, good practice to add a comment. E.g. `// flag = 1 means not seen -1 till now`  
`// flag = 0 means have seen a -1`

Warning: need else here since I will not skip these statements even if `num == -1`

**CRITICAL:** Always initialize your flags



# Avoiding Break using Flags

12

Read integers till you get -1 and print their sum

```
int num, sum = 0;
while(1){
    scanf("%d", &num);
    if(num == -1) break;
    sum += num;
```

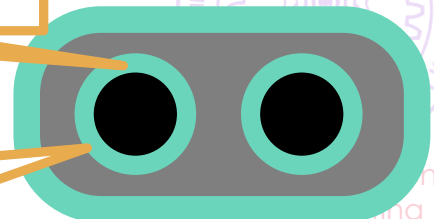
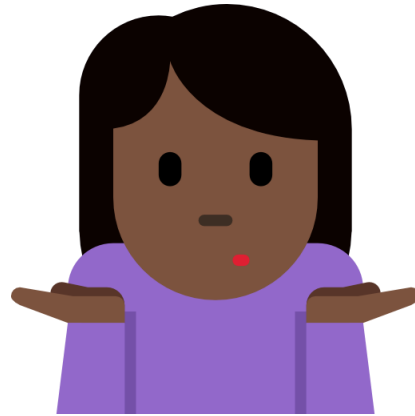
```
int num, sum = 0, flag = 1;
while(flag){
    scanf("%d", &num);
    if(num == -1) flag = 0;
    else sum += num;
```

To avoid confusion or your yourself forgetting what flag values mean, good practice to add a comment. E.g.

```
// flag = 1 means not seen -1 till now
// flag = 0 means have seen a -1
```

Warning: need else here since I will not skip these statements even if num == -1

**CRITICAL:** Always initialize your flags



# Avoiding Bre

Read integers till you

How do I decide how to initialize my flag?  
Last program you set flag = 0 as initial value. Here you set flag = 1 as initial value.

```
int num, sum = 0;
while(1){
    scanf("%d", &num);
    if(num == -1) break;
    sum += num;
```

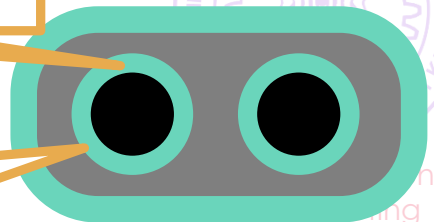
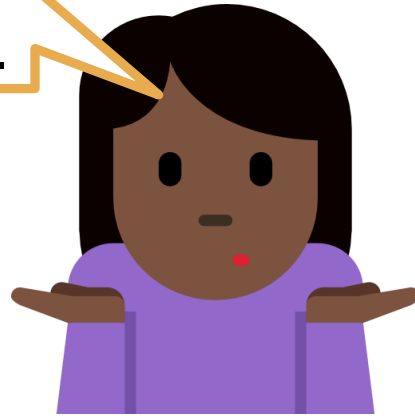
```
int num, sum = 0, flag = 1;
while(flag){
    scanf("%d", &num);
    if(num == -1) flag = 0;
    else sum += num;
```

To avoid confusion or your yourself forgetting what flag values mean, good practice to add a comment. E.g.

```
// flag = 1 means not seen -1 till now
// flag = 0 means have seen a -1
```

Warning: need else here since I will not skip these statements even if num == -1

**CRITICAL:** Always initialize your flags



# Avoiding Bre

Read integers till you

How do I decide how to initialize my flag?  
Last program you set flag = 0 as initial value. Here you set flag = 1 as initial value.

```
int num, sum = 0;
while(1){
    scanf("%d", &num);
    if(num == -1) break;
    sum += num;
```

```
int num, sum = 0, flag = 1;
while(flag){
    scanf("%d", &num);
    if(num == -1) flag = 0;
    else sum += num;
```

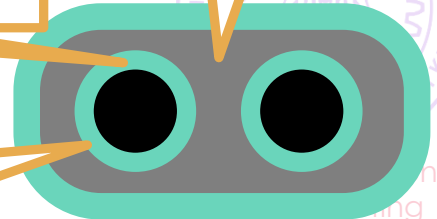
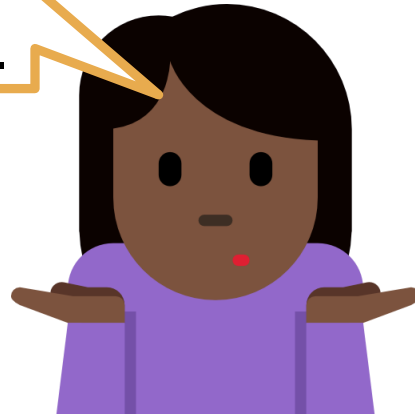
To avoid confusion or your yourself forgetting what flag values mean, good practice to add a comment. E.g.

```
// flag = 1 means not seen -1 till now
// flag = 0 means have seen a -1
```

Warning: need else here since I will not skip these statements even if num == -1

**CRITICAL:** Always initialize your flags

Let us take a few examples – will come with practice



# Non-decreasing Sequences

13



# Non-decreasing Sequences

13

*Keep reading positive numbers till encounter -1 and print YES if the numbers seen so far form a non-decreasing sequence else print NO (Tutorial Problem)*



# Non-decreasing Sequences

13

*Keep reading positive numbers till encounter -1 and print YES if the numbers seen so far form a non-decreasing sequence else print NO (Tutorial Problem)*

**Cases to be considered:**



# Non-decreasing Sequences

13

*Keep reading positive numbers till encounter -1 and print YES if the numbers seen so far form a non-decreasing sequence else print NO (Tutorial Problem)*

## **Cases to be considered:**

Regular (YES): 1 2 3 4 5 6 7 8 -1



# Non-decreasing Sequences

13

*Keep reading positive numbers till encounter -1 and print YES if the numbers seen so far form a non-decreasing sequence else print NO (Tutorial Problem)*

## **Cases to be considered:**

Regular (YES): 1 2 3 4 5 6 7 8 -1

Regular (NO): 1 2 3 4 5 2 3 4 5 -1



# Non-decreasing Sequences

13

*Keep reading positive numbers till encounter -1 and print YES if the numbers seen so far form a non-decreasing sequence else print NO (Tutorial Problem)*

## **Cases to be considered:**

Regular (YES): 1 2 3 4 5 6 7 8 -1

Regular (NO): 1 2 3 4 5 2 3 4 5 -1

Empty stream: -1 (YES by default)



# Non-decreasing Sequences

13

*Keep reading positive numbers till encounter -1 and print YES if the numbers seen so far form a non-decreasing sequence else print NO (Tutorial Problem)*

## **Cases to be considered:**

Regular (YES): 1 2 3 4 5 6 7 8 -1

Regular (NO): 1 2 3 4 5 2 3 4 5 -1

Empty stream: -1 (YES by default)

Singleton stream: 2 -1 (YES by default)



# Factory Output Review

14



# Factory Output Review

14

*Take a number  $n$  and then read  $n$  numbers that indicate output of factory for  $n$  days. If any output is less than 100, print LESS OUTPUT. If all outputs are greater than 200. Print SUPERB OUTPUT.*



# Factory Output Review

14

*Take a number  $n$  and then read  $n$  numbers that indicate output of factory for  $n$  days. If any output is less than 100, print LESS OUTPUT. If all outputs are greater than 200. Print SUPERB OUTPUT.*

**Tip:** If you want to repeat a task  $N$  number of times

```
for(i = 0; i < N; i++){ ... }
```

Or else

```
for(i = 1; i <= N; i++){ ... }
```

**Tip:** use either based on preference, choice, style

Sometimes one more convenient than other



# Responsible use of Flags

15



# Responsible use of Flags

15

Remember, flags are not a new datatype etc. They are just variables that **we**, as programmers chose to treat specially – can have double flags too but not used often



# Responsible use of Flags

15

Remember, flags are not a new datatype etc. They are just variables that **we**, as programmers chose to treat specially – can have double flags too but not used often. To Mr. C, flag variables look just like any other variables.



# Responsible use of Flags

15

Remember, flags are not a new datatype etc. They are just variables that **we**, as programmers chose to treat specially – can have double flags too but not used often

To Mr. C, flag variables look just like any other variables.

If we decide to set an int flag to only 0 or 1, we have to exercise self restraint to not set it to any other value.



# Responsible use of Flags

15

Remember, flags are not a new datatype etc. They are just variables that **we**, as programmers chose to treat specially – can have double flags too but not used often

To Mr. C, flag variables look just like any other variables.

If we decide to set an int flag to only 0 or 1, we have to exercise self restraint to not set it to any other value.

When using flags, also have to be careful about if-else conditions etc.



# Responsible use of Flags

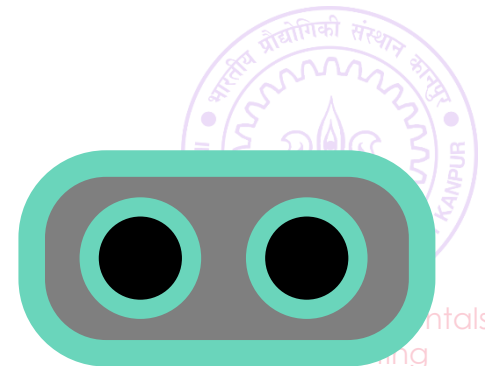
15

Remember, flags are not a new datatype etc. They are just variables that **we**, as programmers chose to treat specially – can have double flags too but not used often

To Mr. C, flag variables look just like any other variables.

If we decide to set an int flag to only 0 or 1, we have to exercise self restraint to not set it to any other value.

When using flags, also have to be careful about if-else conditions etc.



# Responsible use of Flags

15

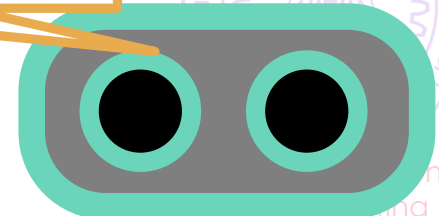
Remember, flags are not a new datatype etc. They are just variables that **we**, as programmers chose to treat specially – can have double flags too but not used often

To Mr. C, flag variables look just like any other variables.

If we decide to set an int flag to only 0 or 1, we have to exercise self restraint to not set it to any other value.

When using flags, also have to be careful about if-else conditions etc.

To me, an int flag variable is just another int variable. I will not warn you if you say `flag = 3`



# Responsible use of Flags

15

Remember, flags are not a new datatype etc. They are just variables that **we**, as programmers chose to treat specially – can have double flags too but not used often

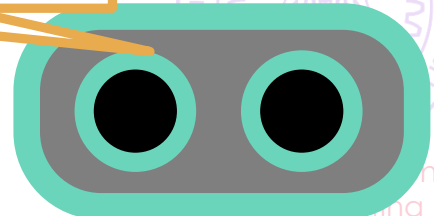
To Mr. C, flag variables look just like any other variables.

If we decide to set an int flag to only 0 or 1, we have to exercise self restraint to not set it to any other value.

When using flags, also have to be careful about if-else conditions etc.

Benefit of flags: clean code

To me, an int flag variable is just another int variable. I will not warn you if you say flag = 3



# Responsible use of Flags

15

Remember, flags are not a new datatype etc. They are just variables that **we**, as programmers chose to treat specially – can have double flags too but not used often

To Mr. C, flag variables look just like any other variables.

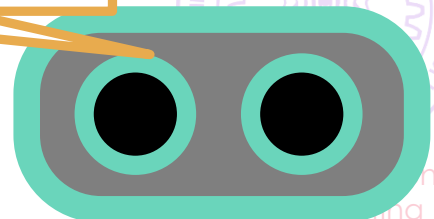
If we decide to set an int flag to only 0 or 1, we have to exercise self restraint to not set it to any other value.

When using flags, also have to be careful about if-else conditions etc.

Benefit of flags: clean code

Too many flags a bad idea!

To me, an int flag variable is just another int variable. I will not warn you if you say flag = 3



# Responsible use of Flags

16



# Responsible use of Flags

16

Flags are better than break or continue since you can always print the value of the flag to find what is going on.



# Responsible use of Flags

16

Flags are better than break or continue since you can always print the value of the flag to find what is going on.

With Break and Continue, easy to get confused why is program breaking or why is it skipping portions of code



# Responsible use of Flags

16

Flags are better than break or continue since you can always print the value of the flag to find what is going on.

With Break and Continue, easy to get confused why is program breaking or why is it skipping portions of code

Overuse of flags bad as well – frowned upon in industry



# Responsible use of Flags

16

Flags are better than break or continue since you can always print the value of the flag to find what is going on.

With Break and Continue, easy to get confused why is program breaking or why is it skipping portions of code

Overuse of flags bad as well – frowned upon in industry

If you have 15 flags, badly named, after few months you will forget why you set those flags – equally bad as break



# Responsible use of Flags

16

Flags are better than break or continue since you can always print the value of the flag to find what is going on.

With Break and Continue, easy to get confused why is program breaking or why is it skipping portions of code

Overuse of flags bad as well – frowned upon in industry

If you have 15 flags, badly named, after few months you will forget why you set those flags – equally bad as break

Especially since flags are usually just set to 0/1



# Responsible use of Flags

16

Flags are better than break or continue since you can always print the value of the flag to find what is going on.

With Break and Continue, easy to get confused why is program breaking or why is it skipping portions of code

Overuse of flags bad as well – frowned upon in industry

If you have 15 flags, badly named, after few months you will forget why you set those flags – equally bad as break

Especially since flags are usually just set to 0/1

You yourself will forget what is flag 14 = 0 or flag 6 = 1 supposed to mean



# Responsible use of Flags

16

Flags are better than break or continue since you can always print the value of the flag to find what is going on.

With Break and Continue, easy to get confused why is program breaking or why is it skipping portions of code

Overuse of flags bad as well – frowned upon in industry

If you have 15 flags, badly named, after few months you will forget why you set those flags – equally bad as break

Especially since flags are usually just set to 0/1

You yourself will forget what is flag 14 = 0 or flag 6 = 1 supposed to mean

A better alternative, use enumerations



# The C Enumeration

17



# The C Enumeration

17

A convenient way to give names to constants



# The C Enumeration

17

A convenient way to give names to constants

Don't have to remember what does `flag = 0` mean



# The C Enumeration

17

A convenient way to give names to constants

Don't have to remember what does `flag = 0` mean

```
enum {Neg, Pos};
```



# The C Enumeration

17

A convenient way to give names to constants

Don't have to remember what does `flag = 0` mean

```
enum {Neg, Pos};
```

Must be valid variable  
names (identifiers)

From now on, Mr C will  
consider

Neg = 0 and Pos = 1  
as **constants**



# The C Enumeration

17

A convenient way to give names to constants

Don't have to remember what does `flag = 0` mean

```
enum {Neg, Pos};
```

Must be valid variable names (identifiers)  
From now on, Mr C will consider  
`Neg = 0` and `Pos = 1`  
as **constants**

Cannot do `Neg++` or  
else `Pos = 2`;



# The C Enumeration

17

A convenient way to give names to constants

Don't have to remember what does `flag = 0` mean

`enum {Neg, Pos};`

Must be valid variable names (identifiers)

From now on, Mr C will consider

`Neg = 0` and `Pos = 1`  
as **constants**

Cannot do `Neg++` or  
else `Pos = 2;`

```
int sum = 0, i, num, flag;
for(i = 1; i <= 100; i++){
    scanf("%d", &num);
    if (num < 0) flag = 0;
    else flag = 1;
    if(flag) sum += num;
}
```



# The C Enumeration

17

A convenient way to give names to constants

Don't have to remember what does `flag = 0` mean

`enum {Neg, Pos};`

Must be valid variable names (identifiers)  
From now on, Mr C will consider  
`Neg = 0` and `Pos = 1`  
as **constants**

Cannot do `Neg++` or  
`else Pos = 2;`

```
int sum = 0, i, num, flag;
for(i = 1; i <= 100; i++){
    scanf("%d", &num);
    if (num < 0) flag = 0;
    else flag = 1;
    if(flag) sum += num;
}
```

```
int sum = 0, i, num, flag;
for(i = 1; i <= 100; i++){
    scanf("%d", &num);
    if (num < 0) flag = Neg;
    else flag = Pos;
    if(flag == Pos)
        sum += num;
}
```

# The C Enumeration

18



# The C Enumeration

18

Bridges the gap between what Mr C is comfortable with  
and what we humans are comfortable with



# The C Enumeration

18

Bridges the gap between what Mr C is comfortable with  
and what we humans are comfortable with

Mr C understands numbers 0, 1 very well



# The C Enumeration

18

Bridges the gap between what Mr C is comfortable with  
and what we humans are comfortable with

Mr C understands numbers 0, 1 very well

Humans understand words TRUE, FALSE, YES, NO better



# The C Enumeration

18

Bridges the gap between what Mr C is comfortable with  
and what we humans are comfortable with

Mr C understands numbers 0, 1 very well

Humans understand words TRUE, FALSE, YES, NO better

Enumerations allow us to link human names to values

