Fun Facts about Functions with Mr C

ESC101: Fundamentals of Computing

Purushottam Kar

Announcements

- Last date for dropping Advanced Track October 12
  - Application must be an email to instructor, mentors, teammates
- Last date for dropping ESC101 course October 12
  - Application must be on standard DoAA course drop form – no email!
- Joint tutorial for B1 and B14 on October 12
  - 12 – 1 PM (same time), L19 - just an arrangement for this week ☺

Mr C takes a Math Lesson

Mr C takes a Math Lesson

Mathematics is full of functions - we define more powerful functions using simple functions

Mr C takes a Math Lesson

Mathematics is full of functions - we define more powerful functions using simple functions

$$\tan(x) = \frac{\sin(x)}{\cos(x)}$$

Mr C takes a Math Lesson

Mathematics is full of functions - we define more powerful functions using simple functions

$$\tan(x) = \frac{\sin(x)}{\cos(x)}$$

sin(x) itself can be defined w.r.t addition, factorial, division

Mr C takes a Math Lesson

Mathematics is full of functions - we define more powerful functions using simple functions

$$\tan(x) = \frac{\sin(x)}{\cos(x)}$$

sin(x) itself can be defined w.r.t addition, factorial, division

$$\sin(x) = x - \frac{x^3}{3!} + \frac{x^5}{5!} - \frac{x^7}{7!} + \dots$$

Mr C takes a Math Lesson

Mathematics is full of functions - we define more powerful functions using simple functions

$$\tan(x) = \frac{\sin(x)}{\cos(x)}$$

sin(x) itself can be defined w.r.t addition, factorial, division

$$\sin(x) = x - \frac{x^3}{3!} + \frac{x^5}{5!} - \frac{x^7}{7!} + \dots$$

Factorial can be defined in terms of multiplication ☺

Mr C takes a Math Lesson

Mathematics is full of functions - we define more powerful functions using simple functions

$$\tan(x) = \frac{\sin(x)}{\cos(x)}$$

sin(x) itself can be defined w.r.t addition, factorial, division

$$\sin(x) = x - \frac{x^3}{3!} + \frac{x^5}{5!} - \frac{x^7}{7!} + \dots$$

Factorial can be defined in terms of multiplication ☺

Multiplication can be defined in terms of addition ☺

Mr C takes a Math Lesson

Mathematics is full of functions - we define more p
functions using simple functions

$$\tan(x) = \frac{\sin(x)}{\cos(x)}$$

sin(x) itself can be defined w.r.t addition, factorial, division

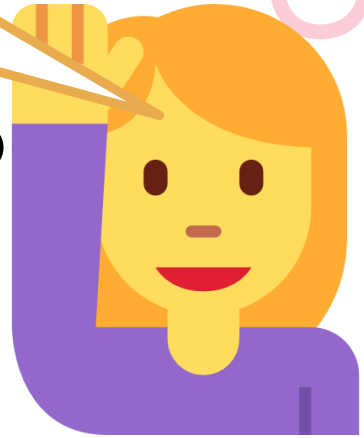
$$\sin(x) = x - \frac{x^3}{3!} + \frac{x^5}{5!} - \frac{x^7}{7!} + \dots$$

Factorial can be defined in terms of multiplication ☺

Multiplication can be defined in terms of addition ☺

Mr C tak

Although we can write tan(x) in terms of addition, subtraction, multiplication and division, we almost never do that. We always write tan(x) = sin(x)/cos(x)



Mathematics is full of functions - we define more p
functions using simple functions

$$\tan(x) = \frac{\sin(x)}{\cos(x)}$$

sin(x) itself can be defined w.r.t addition, factorial, division

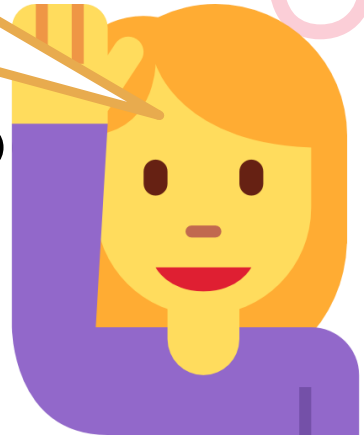
$$\sin(x) = x - \frac{x^3}{3!} + \frac{x^5}{5!} - \frac{x^7}{7!} + \dots$$

Factorial can be defined in terms of multiplication ☺

Multiplication can be defined in terms of addition ☺

Mr C tak

Although we can write tan(x) in terms of addition, subtraction, multiplication and division, we almost never do that. We always write tan(x) = sin(x)/cos(x)



Mathematics is full of functions - we define more p
functions using simple functions

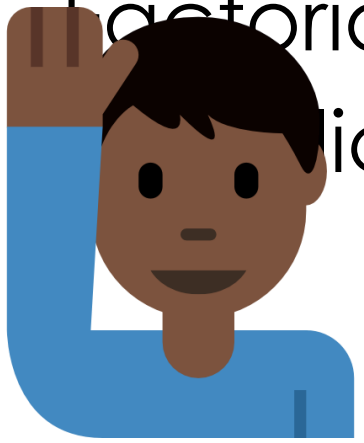
$$\tan(x) = \frac{\sin(x)}{\cos(x)}$$

sin(x) itself can be defined w.r.t addition, factorial, division

$$\sin(x) = x - \frac{x^3}{3!} + \frac{x^5}{5!} - \frac{x^7}{7!} + \dots$$

Factorial can be defined in terms of multiplication ☺

lication can be defined in terms of addition ☺



Mr C tak

Although we can write tan(x) in terms of addition, subtraction, multiplication and division, we almost never do that. We always write tan(x) = sin(x)/cos(x)

Mathematics is full of functions - we define more p
functions using simple functions

$$\tan(x) = \frac{\sin(x)}{\cos(x)}$$

sin(x) itself can be defined w.r.t addition, factorial, division

$$\sin(x) = x - \frac{x^3}{3!} + \frac{x^5}{5!} - \frac{x^7}{7!} + \dots$$

Facto ltiplication ☺

f addition ☺

Helps us write much cleaner math expressions, as well as we do not make mistakes very often

Mr C tak

Although we can write tan(x) in terms of addition, subtraction, multiplication and division, we almost never do that. We always write tan(x) = sin(x)/cos(x)

Mathematics is full of functions - we define more p
functions using simple functions

$$\tan(x) = \frac{\sin(x)}{\cos(x)}$$

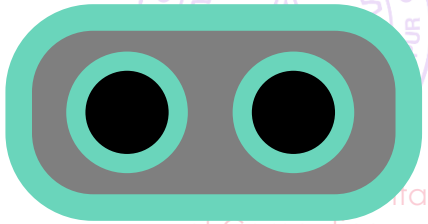
sin(x) itself can be defined w.r.t addition, factorial, division

$$\sin(x) = x - \frac{x^3}{3!} + \frac{x^5}{5!} - \frac{x^7}{7!} + \dots$$

Facto ltiplication ☺

f addition ☺

Helps us write much cleaner math expressions, as well as we do not make mistakes very often



Mr C tak

Although we can write tan(x) in terms of addition, subtraction, multiplication and division, we almost never do that. We always write tan(x) = sin(x)/cos(x)

Mathematics is full of functions - we define more p
functions using simple functions

$$\tan(x) = \frac{\sin(x)}{\cos(x)}$$

sin(x) itself can be defined w.r.t addition, factorial, division

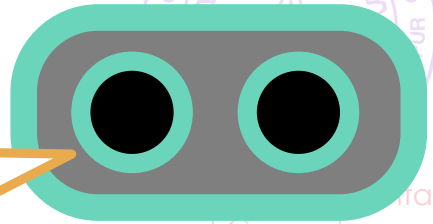
$$\sin(x) = x - \frac{x^3}{3!} + \frac{x^5}{5!} - \frac{x^7}{7!} + \dots$$

Facto ltiplication ☺

f addition ☺

Helps us write much cleaner math expressions, as well as we do not make mistakes very often

I too allow you to write your own functions for your comfort and to make your code easier to read and easier to debug!



The Anatomy of a C Function

The Anatomy of a C Function

HOW WE MUST SPEAK TO MR. COMPILER

The Anatomy of a C Function

HOW WE MUST SPEAK TO MR. COMPILER

The Anatomy of a C Function

HOW WE MUST SPEAK TO MR. COMPILER

```
int isUpperAlpha(char x){



}
```

The Anatomy of a C Function

HOW WE MUST SPEAK TO MR. COMPILER

```
int isUpperAlpha(char x){



}
```

HOW WE USUALLY SPEAK TO A HUMAN

The Anatomy of a C Function

HOW WE MUST SPEAK TO MR. COMPILER

```
int isUpperAlpha(char x){



}
```

HOW WE USUALLY SPEAK TO A HUMAN

The Anatomy of a C Function

HOW WE MUST SPEAK TO MR. COMPILER

```
int isUpperAlpha(char x){



}
```

HOW WE USUALLY SPEAK TO A HUMAN

isUpperAlpha is a function that takes in a character (let us call that character x) as input and gives an integer as output

The Anatomy of a C Function

HOW WE MUST SPEAK TO MR. COMPILER

```
int isUpperAlpha(char x){
    int a = (x >= 'A') && (x <= 'Z');


}
```

HOW WE USUALLY SPEAK TO A HUMAN

isUpperAlpha is a function that takes in a character (let us call that character x) as input and gives an integer as output

The Anatomy of a C Function

HOW WE MUST SPEAK TO MR. COMPILER

```
int isUpperAlpha(char x){
    int a = (x >= 'A') && (x <= 'Z');


}
```

HOW WE USUALLY SPEAK TO A HUMAN

isUpperAlpha is a function that takes in a character (let us call that character x) as input and gives an integer as output

Upon receiving input, please create an integer variable a and store 1 in a if input is upper case alphabet else store 0 in a

The Anatomy of a C Function

HOW WE MUST SPEAK TO MR. COMPILER

```
int isUpperAlpha(char x){
    int a = (x >= 'A') && (x <= 'Z');
    return a;
}
```

HOW WE USUALLY SPEAK TO A HUMAN

isUpperAlpha is a function that takes in a character (let us call that character x) as input and gives an integer as output

Upon receiving input, please create an integer variable a and store 1 in a if input is upper case alphabet else store 0 in a

The Anatomy of a C Function

HOW WE MUST SPEAK TO MR. COMPILER

```
int isUpperAlpha(char x){
    int a = (x >= 'A') && (x <= 'Z');
    return a;
}
```

HOW WE USUALLY SPEAK TO A HUMAN

isUpperAlpha is a function that takes in a character (let us call that character x) as input and gives an integer as output

Upon receiving input, please create an integer variable a and store 1 in a if input is upper case alphabet else store 0 in a

Please output the value of a to whomever used this function

The Anatomy of a C Function

HOW WE MUST SPEAK TO MR. COMPILER

```
int isUpperAlpha(char x){
    int a = (x >= 'A') && (x <= 'Z');
    return a;
}
```

Name of function: isUpperAlpha

HOW WE USUALLY SPEAK TO A HUMAN

isUpperAlpha is a function that takes in a character (let us call that character x) as input and gives an integer as output

Upon receiving input, please create an integer variable a and store 1 in a if input is upper case alphabet else store 0 in a

Please output the value of a to whomever used this function

The Anatomy of a C Function

HOW WE MUST SPEAK TO MR. COMPILER

```
int isUpperAlpha(char x){
    int a = (x >= 'A') && (x <= 'Z');
    return a;
}
```

Name of function: isUpperAlpha

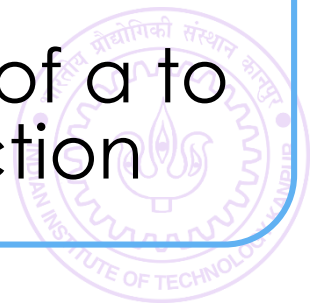
Arguments: one character

HOW WE USUALLY SPEAK TO A HUMAN

isUpperAlpha is a function that takes in a character (let us call that character x) as input and gives an integer as output

Upon receiving input, please create an integer variable a and store 1 in a if input is upper case alphabet else store 0 in a

Please output the value of a to whomever used this function



The Anatomy of a C Function

HOW WE MUST SPEAK TO MR. COMPILER

```
int isUpperAlpha(char x){
    int a = (x >= 'A') && (x <= 'Z');
    return a;
}
```

Name of function: isUpperAlpha

Arguments: one character

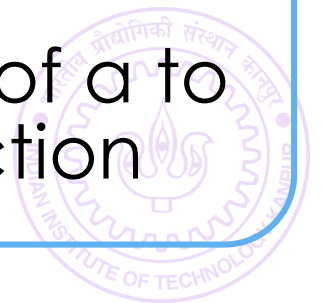
Return type: integer

HOW WE USUALLY SPEAK TO A HUMAN

isUpperAlpha is a function that takes in a character (let us call that character x) as input and gives an integer as output

Upon receiving input, please create an integer variable a and store 1 in a if input is upper case alphabet else store 0 in a

Please output the value of a to whomever used this function



The Anatomy of a C Function

HOW WE MUST SPEAK TO MR. COMPILER

```
int isUpperAlpha(char x){
    int a = (x >= 'A') && (x <= 'Z');
    return a;
}
```

Name of function: isUpperAlpha

Arguments: one character

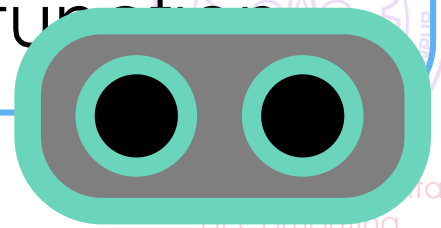
Return type: integer

HOW WE USUALLY SPEAK TO A HUMAN

isUpperAlpha is a function that takes in a character (let us call that character x) as input and gives an integer as output

Upon receiving input, please create an integer variable a and store 1 in a if input is upper case alphabet else store 0 in a

Please output the value of a to whomever used this fu



The Anatomy of a C Function

HOW WE MUST SPEAK TO MR. COMPILER

```
int isUpperAlpha(char x){
    int a = (x >= 'A') && (x <= 'Z');
    return a;
}
```

Name of function: isUpperAlpha

Arguments: one cha

Return type: integer

HOW WE USUALLY SPEAK TO A HUMAN

isUpperAlpha is a function that takes in a character (let us call that character x) as input and gives an integer as output

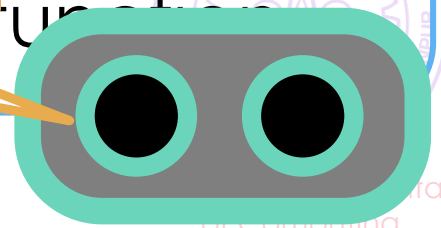
Upon receiving input, please create an integer variable a and store 1 in a if input is upper case alphabet else store 0 in a

lue of a to

his fu

Inputs to a function are called its *arguments*

A function *returns* its output



The Anatomy of a C Function

HOW WE MUST SPEAK TO MR. COMPILER

```
int isUpperAlpha(char x){
    int a = (x >= 'A') && (x <= 'Z');
    return a;
}
```

HOW WE USUALLY SPEAK TO A HUMAN

isUpperAlpha is a function that takes in a character (let us call that character x) as input and gives an integer as output

Upon receiving input, please er variable a if input is upper else store 0 in a lue of a to

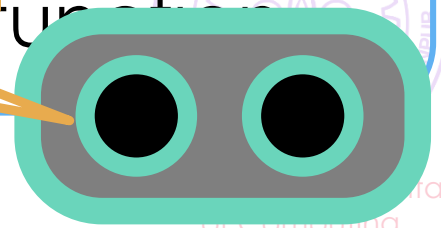
Name of

Arguments: one cha

Return type: integer

Programmers often call the process of giving inputs to a function as *passing arguments to the function*

Inputs to a function are called its *arguments*
A function *returns* its output



The Anatomy of a C Function

HOW WE MUST SPEAK TO MR. COMPILER

```
int isUpperAlpha(char x){
    int a = (x >= 'A') && (x <= 'Z');
    return a;
}
```

HOW WE USUALLY SPEAK TO A HUMAN

isUpperAlpha is a function that takes in a character (let us call that character x) as input and gives an integer as output

Upon receiving input, please er variable a if input is upper else store 0 in a lue of a to

Name of

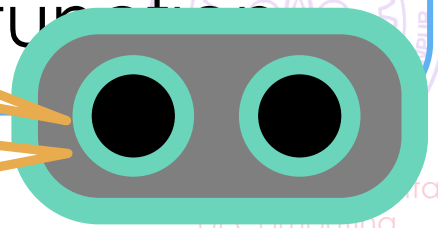
Arguments: one cha

Return type: integer

Programmers often call the process of giving inputs to a function as *passing arguments to the function*

Inputs to a function are called its *arguments*
A function *returns* its output

A function may have many inputs but only one output



The Anatomy of a C Function

HOW WE MUST SPEAK TO MR. COMPILER

```
int isUpperAlpha(char x){
    int a = (x >= 'A') && (x <= 'Z');
    return a;
}
```

HOW WE USUALLY SPEAK TO A HUMAN

isUpperAlpha is a function that takes in a character (let us call that character x) as input and gives an integer as output

Upon receiving input, please er variable a if input is upper else store 0 in a lue of a to

Name of

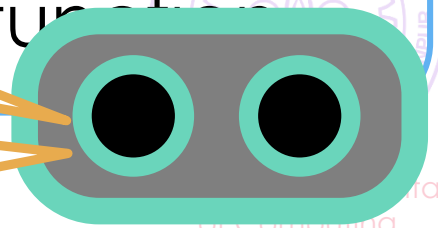
ments: one cha

n type: integer

Programmers often call the process of giving inputs to a function as *passing arguments to the function*

Inputs to a function are called its *arguments*
A function *returns* its output

A function may have many inputs but only one output



The Anatomy of a C Function

HOW WE MUST SPEAK TO MR. COMPILER

```
int isUpperAlpha(char x){
    int a = (x >= 'A') && (x <= 'Z');
    return a;
}
```

HOW WE USUALLY SPEAK TO A HUMAN

isUpperAlpha is a function that takes in a character (let us call that character x) as input and gives an integer as output

Upon receiving input, please
er variable a
if input is upper
else store 0 in a

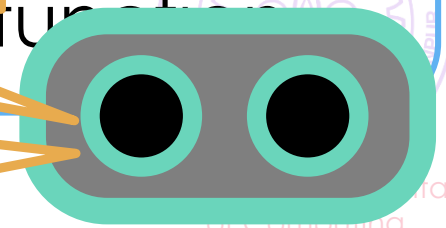
lue of a to

Programmers often call the process of giving inputs to a function as *passing arguments to the function*

Inputs to a function are called its *arguments*
A function *returns* its output

A function may have many inputs but only one output

☹ So I cant write a function that returns 2 integers – say x and y coordinates?



The Anatomy of a C Function

HOW WE MUST SPEAK TO MR. COMPILER

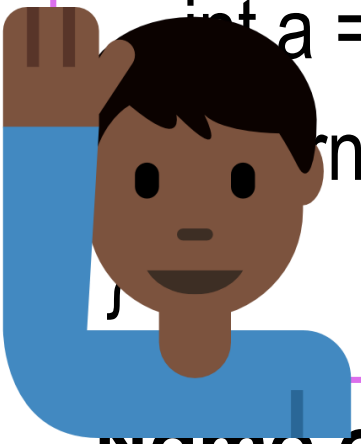
```
int isUpperAlpha(char x){
    int a = (x >= 'A') && (x <= 'Z');
    return a;
}
```

HOW WE USUALLY SPEAK TO A HUMAN

isUpperAlpha is a function that takes in a character (let us call that character x) as input and gives an integer as output

Upon receiving input, please store in integer variable a 1 if input is upper case alphabet else store 0 in a

Return the value of a to whoever calls this function

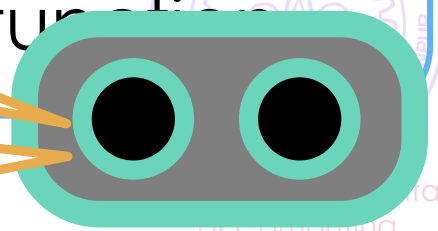


Programmers often call the process of giving inputs to a function as *passing arguments to the function*

Inputs to a function are called its *arguments*
A function *returns* its output

☹ So I cant write a function that returns 2 integers – say x and y coordinates?

A function may have many inputs but only one output



The Anatomy of a C Function

HOW WE MUST SPEAK TO MR. COMPILER

int isU

n a;

Name of

ments: one cho

HOW WE USUALLY SPEAK TO A HUMAN

isUpperAlpha is a function that takes in a character (let us call that character x) as input and gives an integer as output

Upon receiving input, please er variable a if input is upper else store 0 in a

lue of a to

Yes you can! But you have to be a bit clever about doing so

Programmers often call the process of giving inputs to a function as *passing arguments to the function*

Inputs to a function are called its *arguments*
A function *returns* its output

☹ So I cant write a function that returns 2 integers – say x and y coordinates?

A function may have many inputs but only one output

The Anatomy of a C Function

HOW WE MUST SPEAK TO MR. COMPILER

HOW WE USUALLY SPEAK TO A HUMAN

isUpperAlpha is a function that takes in a character (let us call that character x) as input and
nteger as output
eiving input, please
er variable a
if input is upper
else store 0 in a
lue of a to

Yes you can! But you have to be a bit clever about doing so

We will teach you 3 ways to return more than one output in this course ☺

Programmers often call the process of giving inputs to a function as *passing arguments to the function*

Inputs to a function are called its *arguments*
A function *returns* its output

☹ So I cant write a function that returns 2 integers – say x and y coordinates?

A function may have many inputs but only one output

Functional Terminology

Functional Terminology

Function Name: must be a valid identifier abc, a124, _ab1

Functional Terminology

Function Name: must be a valid identifier abc, a124, _ab1

Arguments: can be int, long, float, double, char

Functional Terminology

Function Name: must be a valid identifier abc, a124, _ab1

Arguments: can be int, long, float, double, char

Can also have pointers and even arrays as input – soon!

Functional Terminology

Function Name: must be a valid identifier abc, a124, _ab1

Arguments: can be int, long, float, double, char

Can also have pointers and even arrays as input – soon!

Return type: what does the function *return*

Functional Terminology

Function Name: must be a valid identifier abc, a124, _ab1

Arguments: can be int, long, float, double, char

Can also have pointers and even arrays as input – soon!

Return type: what does the function *return*

When you use a function, we say you have *called* that function. If the function outputs something, we say the function *returned* that output back to you

Functional Terminology

Function Name: must be a valid identifier abc, a124, _ab1

Arguments: can be int, long, float, double, char

Can also have pointers and even arrays as input – soon!

Return type: what does the function *return*

When you use a function, we say you have *called* that function. If the function outputs something, we say the function *returned* that output back to you

The English word return has two meanings
I returned from the ESC101 lab at 5PM
I returned the two books I had issued from the library

Functional Terminology

Function Name: must be a valid identifier abc, a124, _ab1

Arguments: can be int, long, float, double, char

Can also have pointers and even arrays as input – soon!

Return type: what does the function *return*

When you use a function, we say you have *called* that function. If the function outputs something, we say the function *returned* that output back to you

The English word return has two meanings
I returned from the ESC101 lab at 5PM
I returned the two books I had issued from the library

Functions return back values to you just as you return books

Functional Terminology

Function Name: must be a valid identifier abc, a124, _ab1

Arguments: can be int, long, float, double, char

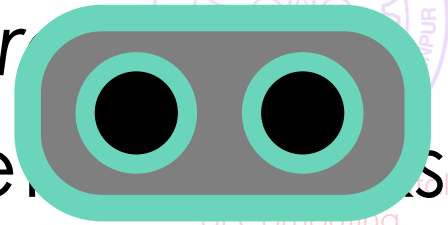
Can also have pointers and even arrays as input – soon!

Return type: what does the function *return*

When you use a function, we say you have *called* that function. If the function outputs something, we say the function *returned* that output back to you

The English word return has two meanings
I returned from the ESC101 lab at 5PM
I returned the two books I had issued from the libr

Functions return back values to you just as you re



Functional Terminology

Function Name: must be a valid identifier abc, a124, _ab1

Arguments: can be int, long, float, double, char

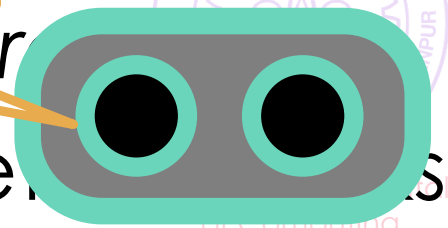
Can also have pointers and even arrays as input – soon!

Return type: what does the function *return*

When you use a function, we say you have *called* that function. If the function outputs something, we say the function *returned* that output back to you

The English word
I returned from
I returned the two books I had issued from

You must define the function **before** using the function (within main or your own functions)



Functions return back values to you just as you re

Some Functional Exercises

Some Functional Exercises

Define a function to input two integers, output their max

Some Functional Exercises

Define a function to input two integers, output their max

Define a function to print Hello World

Some Functional Exercises

Define a function to input two integers, output their max

Define a function to print Hello World

Define a function to output 1 if input is prime else 0

Some Functional Exercises

Define a function to input two integers, output their max

Define a function to print Hello World

Define a function to output 1 if input is prime else 0

Define a function to input two integers and print Hello World if their max is prime

Some Functional Exercises

Define a function to input two integers, output their max

Define a function to print Hello World

Define a function to output 1 if input is prime else 0

Define a function to input two integers and print Hello World if their max is prime

Define a function to print the max of 3 numbers

Some Functional Exercises

Define a function to input two integers, output their max

Define a function to print Hello World

Define a function to output 1 if input is prime else 0

Define a function to input two integers and print Hello World if their max is prime

Define a function to print the max of 3 numbers

Define a function to input a character, output its upper case version if lower case else output the character itself

Arguments and Return types

Arguments and Return types

You can define a function that takes in no input and gives no output

Arguments and Return types

You can define a function that takes in no input and gives no output

```
void print(void){
    printf("Hello World");
}
```

Arguments and Return types

You can define a function that takes in no input and gives no output

Even void print(){ … } works

```
void print(void){
    printf("Hello World");
}
```

Arguments and Return types

You can define a function that takes in no input and gives no output

Even void print(){ … } works

```
void print(void){
    printf("Hello World");
}
```

You can define a function that takes inputs but gives no output

Arguments and Return types

You can define a function that takes in no input and gives no output

Even void print(){ … } works

```
void print(void){
    printf("Hello World");
}
```

You can define a function that takes inputs but gives no output

```
void sum(int a, int b){
    printf("Sum %d", a+b);
}
```

Arguments and Return types

You can define a function that takes in no input and gives no output

Even void print(){ … } works

```
void print(void){
    printf("Hello World");
}
```

You can define a function that takes inputs but gives no output

```
void sum(int a, int b){
    printf("Sum %d", a+b);
}
```

You can define a function that takes no input but gives an output

Arguments and Return types

You can define a function that takes in no input and gives no output

Even void print(){ … } works

```
void print(void){
    printf("Hello World");
}
```

You can define a function that takes inputs but gives no output

```
void sum(int a, int b){
    printf("Sum %d", a+b);
}
```

You can define a function that takes no input but gives an output

```
char getFirstAlpha(void){
    return 'A';
}
```

Arguments and Return types

You can define a function that takes in no input and gives no output

Even void print(){ … } works

```
void print(void){
    printf("Hello World");
}
```

You can define a function that takes inputs but gives no output

```
void sum(int a, int b){
    printf("Sum %d", a+b);
}
```

You can define a function that takes no input but gives an output

Even char getFirstAlpha(){ … } works

```
char getFirstAlpha(void){
    return 'A';
}
```

More on Arguments

More on Arguments

The names you give to the input variables of a function when writing a function can be any valid identifiers

More on Arguments

The names you give to the input variables of a function when writing a function can be any valid identifiers

They can even be variable names you are using in other functions e.g. inside the main() function

More on Arguments

The names you give to the input variables of a function when writing a function can be any valid identifiers

They can even be variable names you are using in other functions e.g. inside the main() function

Warning: do not expect Mr C to automatically copy values from one function to another just because two variable have the same name!

More on Arguments

The names you give to the input variables of a function when writing a function can be any valid identifiers

They can even be variable names you are using in other functions e.g. inside the main() function

Warning: do not expect Mr C to automatically copy values from one function to another just because two variable have the same name!

Calling a function is like creating a clone of Mr C. This clone starts afresh, with any inputs you have given. The clone forgets all old variable names and values.

More on Arguments

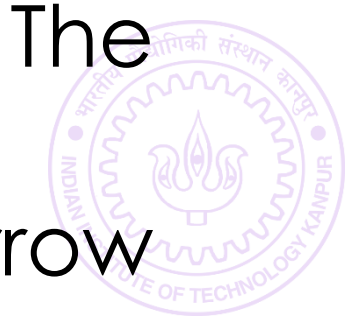
The names you give to the input variables of a function when writing a function can be any valid identifiers

They can even be variable names you are using in other functions e.g. inside the main() function

Warning: do not expect Mr C to automatically copy values from one function to another just because two variable have the same name!

Calling a function is like creating a clone of Mr C. This clone starts afresh, with any inputs you have given. The clone forgets all old variable names and values.

Will see more about this "cloning" behaviour tomorrow



More on Arguments

More on Arguments

If you have promised to give a function two integers, please give it two integers

More on Arguments

If you have promised to give a function two integers, please give it two integers

If you give it only one or three integers, compilation error

More on Arguments

If you have promised to give a function two integers, please give it two integers

If you give it only one or three integers, compilation error

If you give it two floats or else one char and one int, automatic typecasting will take place

More on Arguments

If you have promised to give a function two integers, please give it two integers

If you give it only one or three integers, compilation error

If you give it two floats or else one char and one int, automatic typecasting will take place

Be careful to not make typecasting errors

More on Return

More on Return

May write return statement many times inside a function

More on Return

May write return statement many times inside a function

When Mr C (his clone actually) sees a return statement, he immediately generates the output and function execution stops there.

More on Return

May write return statement many times inside a function

When Mr C (his clone actually) sees a return statement, he immediately generates the output and function execution stops there.

The clone dies and the original Mr C takes over ☺

More on Return

May write return statement many times inside a function

When Mr C (his clone actually) sees a return statement, he immediately generates the output and function execution stops there.

The clone dies and the original Mr C takes over ☺

Warning: if you have promised that a function returns an integer, all return statements in that function must return an integer value – otherwise compilation error!

More on Return

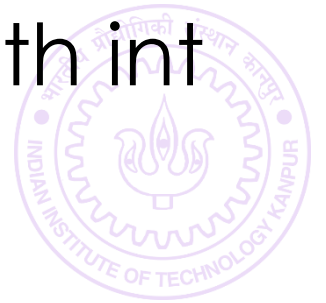
May write return statement many times inside a function

When Mr C (his clone actually) sees a return statement, he immediately generates the output and function execution stops there.

The clone dies and the original Mr C takes over ☺

Warning: if you have promised that a function returns an integer, all return statements in that function must return an integer value – otherwise compilation error!

If you return a float/double value from a function with int return type, automatic typecasting will take place.



More on Return

May write return statement many times inside a function

When Mr C (his clone actually) sees a return statement, he immediately generates the output and function execution stops there.

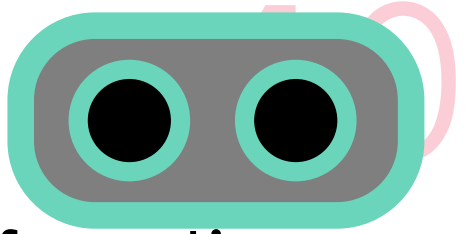
The clone dies and the original Mr C takes over ☺

Warning: if you have promised that a function returns an integer, all return statements in that function must return an integer value – otherwise compilation error!

If you return a float/double value from a function with int return type, automatic typecasting will take place.

Be careful to not make typecasting mistakes

More on Return



May write return statement many times inside a function

When Mr C (his clone actually) sees a return statement, he immediately generates the output and function execution stops there.

The clone dies and the original Mr C takes over ☺

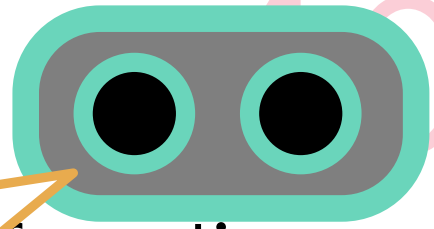
Warning: if you have promised that a function returns an integer, all return statements in that function must return an integer value – otherwise compilation error!

If you return a float/double value from a function with int return type, automatic typecasting will take place.

Be careful to not make typecasting mistakes

More o

For functions that do not need to return anything i.e. void return type, you can either say return; or else not write return at all inside the function body in which case the entire body will get executed



May write re function

When Mr C (his clone actually) sees a return statement, he immediately generates the output and function execution stops there.

The clone dies and the original Mr C takes over ☺

Warning: if you have promised that a function returns an integer, all return statements in that function must return an integer value – otherwise compilation error!

If you return a float/double value from a function with int return type, automatic typecasting will take place.

Be careful to not make typecasting mistakes

More on Return

More on Return

The value that is returned can be used safely just as a normal variable of that same data type

More on Return

The value that is returned can be used safely just as a normal variable of that same data type

You can freely use returned values in expressions

More on Return

The value that is returned can be used safely just as a normal variable of that same data type

You can freely use returned values in expressions

- Be careful of type though

More on Return

The value that is returned can be used safely just as a normal variable of that same data type

You can freely use returned values in expressions

- Be careful of type though

Did you know that the printf function also returns an integer (the number of characters printed) ☺

More on Return

The value that is returned can be used safely just as a normal variable of that same data type

You can freely use returned values in expressions

- Be careful of type though

Did you know that the printf function also returns an integer (the number of characters printed) ☺

scanf() also returns an integer – find out what that is !!

More on Return

The value that is returned can be used safely just as a normal variable of that same data type

You can freely use returned values in expressions

- Be careful of type though

Did you know that the printf function also returns an integer (the number of characters printed) ☺

scanf() also returns an integer – find out what that is !!

```
int sum(int x, int y){
    return x + y;
}
```

More on Return

The value that is returned can be used safely just as a normal variable of that same data type

You can freely use returned values in expressions

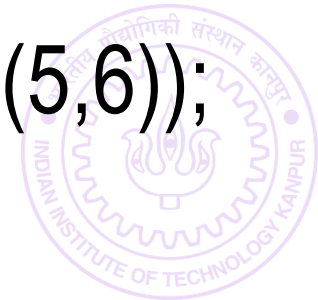
- Be careful of type though

Did you know that the printf function also returns an integer (the number of characters printed) ☺

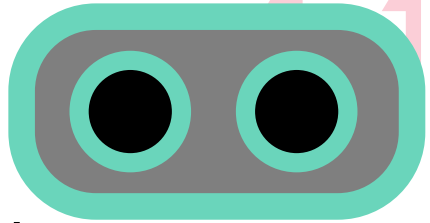
scanf() also returns an integer – find out what that is !!

```
int sum(int x, int y){
    return x + y;
}
```

```
int main(){
    printf("%d", sum(3,4) - sum(5,6));
    return 0;
}
```



More on Return



The value that is returned can be used safely just as a normal variable of that same data type

You can freely use returned values in expressions

- Be careful of type though

Did you know that the printf function also returns an integer (the number of characters printed) ☺

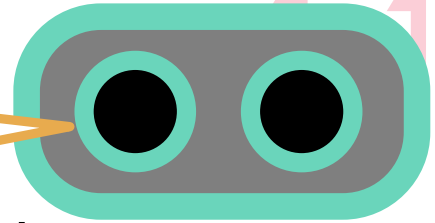
scanf() also returns an integer – find out what that is !!

```
int sum(int x, int y){
    return x + y;
}
```

```
int main(){
    printf("%d", sum(3,4) - sum(5,6));
    return 0;
}
```

More on Return

main() is also a function with return type int



The value that is returned can be used safely just as a normal variable of that same data type

You can freely use returned values in expressions

- Be careful of type though

Did you know that the printf function also returns an integer (the number of characters printed) ☺

scanf() also returns an integer – find out what that is !!

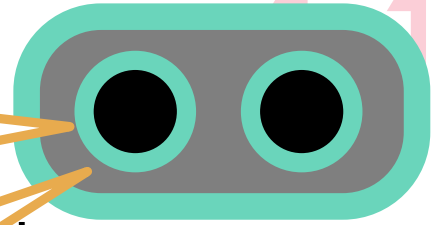
```
int sum(int x, int y){
    return x + y;
}
```

```
int main(){
    printf("%d", sum(3,4) - sum(5,6));
    return 0;
}
```

More on Return

main() is also a function with return type int

main() is like a reserved function name. Cannot name your function main



The value that is re[illegible]st as a normal variable of [illegible]

You can freely use returned values in expressions

- Be careful of type though

Did you know that the printf function also returns an integer (the number of characters printed) ☺

scanf() also returns an integer – find out what that is !!

```
int sum(int x, int y){
    return x + y;
}
```

```
int main(){
    printf("%d", sum(3,4) - sum(5,6));
    return 0;
}
```

Benefits of writing functions

Benefits of writing functions

Allows you to think very clearly

Benefits of writing functions

Allows you to think very clearly

E.g. if you want to do something if the integer n is a prime number or if it is divisible by 11

Benefits of writing functions

Allows you to think very clearly

E.g. if you want to do something if the integer n is a prime number or if it is divisible by 11

```
if(isPrime(n) || isDivby11(n)){
    …
}
```

Benefits of writing functions

Allows you to think very clearly

E.g. if you want to do something if the integer n is a prime number or if it is divisible by 11

```
if(isPrime(n) || isDivby11(n)){
    …
}
```

Write the body of the if condition without worrying about primality testing etc and then define the functions later ☺

Benefits of writing functions

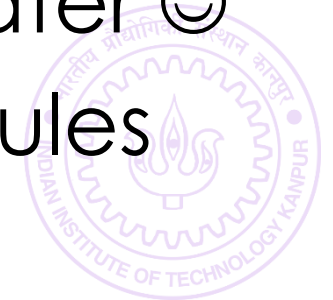
Allows you to think very clearly

E.g. if you want to do something if the integer n is a prime number or if it is divisible by 11

```
if(isPrime(n) || isDivby11(n)){
    …
}
```

Write the body of the if condition without worrying about primality testing etc and then define the functions later ☺

You can break your code into chunks – called modules



Benefits of writing functions

Allows you to think very clearly

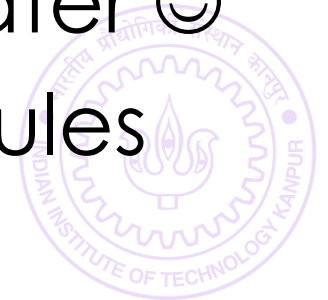
E.g. if you want to do something if the integer n is a prime number or if it is divisible by 11

```
if(isPrime(n) || isDivby11(n)){
    …
}
```

Write the body of the if condition without worrying about primality testing etc and then define the functions later ☺

You can break your code into chunks – called modules

Each module handled using a separate function



Benefits of writing functions

Allows you to think very clearly

E.g. if you want to do something if the integer n is a prime number or if it is divisible by 11

```
if(isPrime(n) || isDivby11(n)){
    …
}
```

Write the body of the if condition without worrying about primality testing etc and then define the functions later ☺

You can break your code into chunks – called modules

Each module handled using a separate function

Benefits of writing functions

Allows you to think very clearly

E.g. if you want to do so… a prime number or if it is divisible …

E.g. in this case, primality testing is one module, checking for divisibility by 11 is another module

```
if(isPrime(n) || isDivby11(n)){
    …
}
```

Write the body of the if condition without worrying about primality testing etc and then define the functions later ☺

You can break your code into chunks – called modules

Each module handled using a separate function

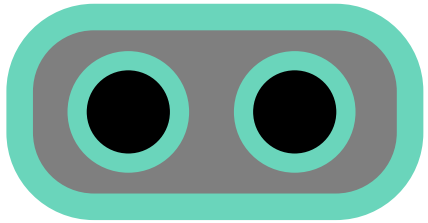
Benefits of writing functions

Allows you to think very clearly

E.g. if you want to do so[illegible] a prime number or if it is divisible

E.g. in this case, primality testing is one module, checking for divisibility by 11 is another module

```
if(isPrime(n) || isDivby11(n)){
    ...
}
```



Write the body of the if condition without worrying about primality testing etc and then define the functions later ☺

You can break your code into chunks – called modules

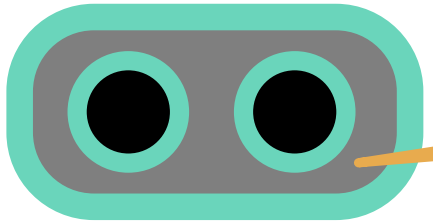
Each module handled using a separate function

Benefits of writing functions

Allows you to think very clearly

E.g. if you want to do so… a prime number or if it is divisible …

```
if(isPrime(n) || isDivby11(n)){
    …
}
```



E.g. in this case, primality testing is one module, checking for divisibility by 11 is another module

Writing code that has modules is a type of *modular programming* – it is the the industry standard!

Write the body of the if condition without worrying about primality testing etc and then define the functions later ☺

You can break your code into chunks – called modules

Each module handled using a separate function

Benefits of writing functions

Benefits of writing functions

Functions allow you to write very neat, readable code

Benefits of writing functions

Functions allow you to write very neat, readable code

Use function names that describe what the function does

Benefits of writing functions

Functions allow you to write very neat, readable code

Use function names that describe what the function does

Your co-workers/team-mates will be able to understand your code much better if it has nice readable functions

Benefits of writing functions

Functions allow you to write very neat, readable code

Use function names that describe what the function does

Your co-workers/team-mates will be able to understand your code much better if it has nice readable functions

Functions allow you to debug your program faster

Benefits of writing functions

Functions allow you to write very neat, readable code

Use function names that describe what the function does

Your co-workers/team-mates will be able to understand your code much better if it has nice readable functions

Functions allow you to debug your program faster

If code is broken into function, to debug, find out which function is not working properly ☺

Benefits of writing functions

Functions allow you to write very neat, readable code

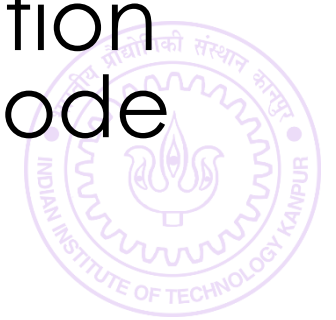
Use function names that describe what the function does

Your co-workers/team-mates will be able to understand your code much better if it has nice readable functions

Functions allow you to debug your program faster

If code is broken into function, to debug, find out which function is not working properly ☺

Rest of code need not be touched, only faulty function needs to be fixed – again the industry standard of code maintenance.



Benefits of writing functions

Benefits of writing functions

Functions allow you to reuse code

Benefits of writing functions

Functions allow you to reuse code

We are so grateful some one wrote functions like sqrt(), abs() in math.h that we are able to use again and again ☺

Benefits of writing functions

Functions allow you to reuse code

We are so grateful some one wrote functions like sqrt(), abs() in math.h that we are able to use again and again ☺

printf() and scanf() are also functions. Think of how much we use them in every single program

Benefits of writing functions

Functions allow you to reuse code

We are so grateful some one wrote functions like sqrt(), abs() in math.h that we are able to use again and again ☺

printf() and scanf() are also functions. Think of how much we use them in every single program

We are reusing code that some helpful C expert wrote in the printf(), scanf(), sqrt(), abs() and other functions

Benefits of writing functions

Functions allow you to reuse code

We are so grateful some one wrote functions like sqrt(), abs() in math.h that we are able to use again and again ☺

printf() and scanf() are also functions. Think of how much we use them in every single program

We are reusing code that some helpful C expert wrote in the printf(), scanf(), sqrt(), abs() and other functions

If some piece of code keeps getting used in your program again and again – put it inside a function!

Benefits of writing functions

Functions allow you to reuse code

We are so grateful some one wrote functions like sqrt(), abs() in math.h that we are able to use again and again ☺

printf() and scanf() are also functions. Think of how much we use them in every single program

We are reusing code that some helpful C expert wrote in the printf(), scanf(), sqrt(), abs() and other functions

If some piece of code keeps getting used in your program again and again – put it inside a function!

We reused code in today's codes – didn't have to rewrite code – may make mistakes if you write same code again