

Functions

Adapted from materials by Dr. Carrier



CIS 162/163 Recap

CIS 162/163 Recap

What is a function?

CIS 162/163 Recap

What is a function?

- Blocks of code that can be called from elsewhere
- Usually accomplish one specific task

CIS 162/163 Recap

What is a function?

- Blocks of code that can be called from elsewhere
- Usually accomplish one specific task

Why would we use them?

CIS 162/163 Recap

What is a function?

- Blocks of code that can be called from elsewhere
- Usually accomplish one specific task

Why would we use them?

- Save time writing code
- Improves readability
- Modularity

Comparing to Python

```
def get_smaller(a, b):  
    if a < b:  
        return a  
    return b;
```

Comparing to Python

```
def get_smaller(a, b):  
    if a < b:  
        return a  
    return b;
```

Roughly the same function in C:

Comparing to Python

```
def get_smaller(a, b):  
    if a < b:  
        return a  
    return b;
```

Roughly the same function in C:

```
int GetSmaller(int a, int b){  
    if(a < b) return a;  
    return b;  
}
```

Comparing to Python

```
def get_smaller(a, b):  
    if a < b:  
        return a  
    return b;
```

Roughly the same function in C:

```
int GetSmaller(int a, int b){  
    if(a < b) return a;  
    return b;  
}
```

Calling a function:

```
int res = GetSmaller(7, x);
```

Terminology

```
int GetSmaller(int a, int b){  
    if(a < b) return a;  
    return b;  
}
```

Calling a function:

```
int res = GetSmaller(7, x);
```

Terminology

Function name



```
int GetSmaller(int a, int b){  
    if(a < b) return a;  
    return b;  
}
```


Calling a function:

```
int res = GetSmaller(7, x);
```

Terminology

Function name

Parameters



```
int GetSmaller(int a, int b){  
    if(a < b) return a;  
    return b;  
}
```


Calling a function:

```
int res = GetSmaller(7, x);
```

Terminology

Function name

Parameters



```
int GetSmaller(int a, int b){  
    if(a < b) return a;  
    return b;  
}
```

Function body

The diagram illustrates the components of a C++ function definition. A red arrow points from the label 'Function name' to the identifier 'GetSmaller' in the code. Another red arrow points from the label 'Parameters' to the parameter list '(int a, int b)'. A third red arrow points from the label 'Function body' to the curly braces '{ ... }' that enclose the function's logic.

Calling a function:

```
int res = GetSmaller(7, x);
```

Terminology

Function name

Parameters

```
int GetSmaller(int a, int b){  
    if(a < b) return a;  
    return b;  
}
```

Function body

Calling a function:

```
int res = GetSmaller(7, x);
```

Arguments

Little more terminology

Function prototype:

```
int GetSmaller(int a, int b);  
    or  
int GetSmaller(int, int);
```


Little more terminology

Function prototype:

```
int GetSmaller(int a, int b);  
    or  
int GetSmaller(int, int);
```

Function definition:

```
int GetSmaller(int a, int b){  
    if(a < b) return a;  
    return b;  
}
```

Little more terminology

Function prototype:

```
int GetSmaller(int a, int b);  
    or  
int GetSmaller(int, int);
```

Function definition:

```
int GetSmaller(int a, int b){  
    if(a < b) return a;  
    return b;  
}
```


Key: Definition can come before or after main

If definition is after main, you *must* have prototype before main

Returning


```
int GetSmaller(int a, int b){  
    if(a < b) return a;  
    return b;  
}
```

Returning **Return type**



```
int GetSmaller(int a, int b){  
    if(a < b) return a;  
    return b;  
}
```


Returning Return type



```
int GetSmaller(int a, int b){  
    if(a < b) return a;  
    return b;  
}
```

All functions must have a return type

Returning Return type



```
int GetSmaller(int a, int b){  
    if(a < b) return a;  
    return b;  
}
```

All functions must have a return type

If you don't return anything, return type is `void`

Arguments

```
int Increment(int x){  
    x++;  
    return x;  
}  
  
int main(){  
    int i = 0;  
    printf("i = %d\n", i);  
    int res = Increment(i);  
    printf("res = %d\n", res);  
    printf("i = %d\n", i);  
}
```

What does this output?

Arguments

```
int Increment(int x){
    x++;
    return x;
}

int main(){
    int i = 0;
    printf("i = %d\n", i);        // 0
    int res = Increment(i);
    printf("res = %d\n", res);    // 1
    printf("i = %d\n", i);        // 0
}
```

What does this output?

Arguments

C is always **pass-by-value**

Not pass-by-reference

Arguments

C is always **pass-by-value**

Not pass-by-reference

How can we change arguments?

Pass a pointer!

Arguments

```
int Increment(int* p){
    (*p)++;
    return *p;
}

int main(){
    int i = 0;
    printf("i = %d\n", i);
    int res = Increment(&i);
    printf("res = %d\n", res);
    printf("i = %d\n", i);
}
```

What does this output?

Arguments

```
int Increment(int* p){
    (*p)++;
    return *p;
}

int main(){
    int i = 0;
    printf("i = %d\n", i);      // 0
    int res = Increment(&i);
    printf("res = %d\n", res);  // 1
    printf("i = %d\n", i);      // 1
}
```

What does this output?

Arguments

What about passing an array?

How can we write a function to print an array in a nice way?

Arguments

What about passing an array?

How can we write a function to print an array in a nice way?

```
void PrintArray(int* arr, int len){  
    printf("[");  
    int i = 0;  
    for(i = 0; i < len; i++){  
        printf(" %d", arr[i]);  
    }  
    printf(" ]\n");  
}
```

Arguments

What about passing an array?

How can we write a function to print an array in a nice way?

```
void PrintArray(int* arr, int len){  
    printf("[");  
    int i = 0;  
    for(i = 0; i < len; i++){  
        printf(" %d", arr[i]);  
    }  
    printf(" ]\n");  
}
```

Notice we also have to pass the length!

Scope

What is the output of this code?

```
void DoStuff(){
    int x = 10;
    printf("In func: x = %d\n", x);
    ...
}

int main(){
    int x = 0;
    printf("x = %d\n", x);
    DoStuff();
    printf("x = %d\n", x);
}
```


Scope

What is the output of this code?

```
void DoStuff(){
    int x = 10;           //v x = 10
    printf("In func: x = %d\n", x);
    ...
}

int main(){
    int x = 0;
    printf("x = %d\n", x); // x = 0
    DoStuff();
    printf("x = %d\n", x); // x = 0
}
```

Scope

Functions have their own scope

Scope

Functions have their own scope

Variables in function don't exist outside it

Functions can't access variables in main

Scope

Functions have their own scope

Variables in function don't exist outside it

Functions can't access variables in main

Is this code valid?

```
int* MakeInt(){  
    int x = 10;  
    return &x;  
}
```

Scope

Functions have their own scope

Variables in function don't exist outside it

Functions can't access variables in main

Is this code valid?

```
int* MakeInt(){  
    int x = 10;  
    return &x;  
}
```

Nope! x falls out of scope when we return

Memory address is then pointing at nothing :(