# From Java to C: Agenda

Syntax

Stack Diagrams

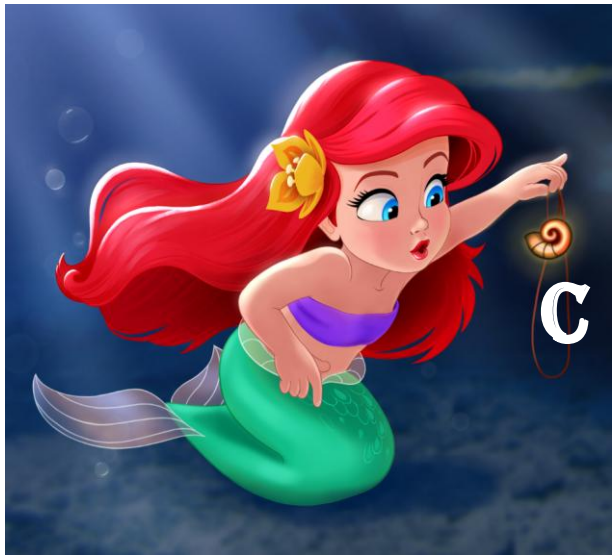
Built-in Types

Console I/O

Arrays

Strings

Structs



# From Java to C

Syntax is very similar!

- if/else statements, loops (while/for), variables, functions
- {} for statement blocks
- ; for statements
- operators and operator precedence the same
  - +, -, *, /, +=, &&, ||, % etc!

Data types

- int (1, -5, 0)
- double, float (1.0, -4.7)
- char (‘a’, ‘$’, ‘\n’)

# The Stack

**Execution stack** – keeps track of active functions

**Stack frame –** created by each function

- contains parameters, local variables
- topmost frame is currently executing function
- frame is pushed to stack when function is called
- frame is popped off stack when it returns

We will use **stack diagrams** to visualize program flow

https://pythontutor.com/c.html for checking your work

# From Java to C

What is the output of this program?

```
int func(int a, int b) {
    a += 5;
    if (a > 10) return 0;
    return a - b;
}

int main() {
    int x, y;  // declare two integers
    x = 4;
    y = 7;
    y = func(x, y);
    for (int i = 0; i < 3; i++) {
      printf("%d, %d\n", x, y);
    }

    return 0;
}
```

*Table 2. C Numeric Types*

| Type name | Usual size | Values stored | How to declare |
|---|---|---|---|
| `char` | 1 byte | integers | `char x;` |
| `short` | 2 bytes | signed integers | `short x;` |
| `int` | 4 bytes | signed integers | `int x;` |
| `long` | 4 or 8 bytes | signed integers | `long x;` |
| `long long` | 8 bytes | signed integers | `long long x;` |
| `float` | 4 bytes | signed real numbers | `float x;` |
| `double` | 8 bytes | signed real numbers | `double x;` |

Java has these too…

Questions:

1. What is a byte?
2. Why might we want to use an `int` instead of a `long`, or `float` instead of `double`?
3. Why might we want to use an `long` instead of a `int`, or `double` instead of `float`?

# printf function

- Same as System.out.printf():

Java: `System.out.printf("%d %s\t %f", 6, "hello", 3.4);`

C: `printf("%d %s\t %f\n", 6, "hello", 3.4);`

| | |
|---|---|
| %d, %i | int |
| %f or %g | float or double |
| %c | char |
| %s | string |
| %lu | Long unsigned int |
| %x | Unsigned hexidecimal |
| \t \n | tab character, new line character |

To add padding: %03d, %.2f

See: https://www.cplusplus.com/reference/cstdio/printf/

# scanf

- For reading in values of different types
- Uses format string like printf
- The arguments are the memory locations into which the values will be stored (the address of program variables or base addr of arrays):

```
int x;
float y;
char s[100];

scanf(" %s%d%f", s, &x, &y);

// s is the base address of the string array
// &x is the address of the variable x in memory
// &y is the address of the variable y in memory
```



Slide by Tia Newhall, CS31, Swarthmore

# scanf: Example

```
int main() {
  printf("Do you like jokes? (y/n): ");

  char response = 0;
  scanf("%c", &response);

  printf("You chose: %c\n", response);
  if (response == 'y') {
    printf("Yes, me too!\n");
  } else if (response == 'n') {
    printf("Me neither!\n");
  } else {
    printf("I don't get you.\n");
  }

  return 0;
}
```

# What about Booleans?

**No booleans in C**

0 is **always false**

Every other value is **true**

# Boolean: Examples

```
#include <stdio.h>

int isEven(int val) {
  if (val % 2 == 0) {
    return 1;
  }
  return 0;
}
```

```
int main() {

  int a = 0;
  int b = 7;

  if (isEven(a)) {
    printf("a is even\n");
  } else {
    printf("a is odd\n");
  }

  if (b) {
    printf("Any non-zero value is true!\n");
  }
  return 0;
}
```

# Arrays

```
class Array {
  public static void main(String[] args) {
    int i, size = 0;
    int[] my_arr = new int[10];

    for (i = 0; i < 10; i++) {
        my_arr[i] = i;
        size++;
    }
   my_arr[3] = 100;

   System.out.printf("array of %d items:\n",
         size);

    for (i = 0; i < 10; i++) {
        System.out.printf("%d\n", my_arr[i]);
    }
  }
}
```

```
int main() {
    int i, size = 0;
    int my_arr[10];

    for (i = 0; i < 10; i++) {
        my_arr[i] = i;
        size++;
    }

    my_arr[3] = 100;
    printf("array of %d items:\n", size);

   for (i = 0; i < 10; i++) {
        printf("%d\n", my_arr[i]);
    }

    return 0;
}
```

# Arrays

**static allocation –** the size of the array is set at compile time *and does not change at runtime*

**dynamic allocation –** the size of the array can be set (and changed) *while the program is running*

# Arrays

- No bounds checking in C!

- Easy to write/read elements out of bounds of array
  - You may (or may) not see a problem depending on what memory you access…

# Invalid Index: Example in C

```
#include <stdio.h>

int main() {
  int bad[4] = {1, 2, 3, 4};
  printf("bad[2] = %d\n", bad[2]);
  printf("bad[4] = %d\n", bad[4]);
  return 0;
}
```

```
WSL alinen@Xin:~/cs223/cs223-devel/chpt01$ ./a.out
bad[2] = 3
bad[4] = -1514561552
```

# Invalid Index: Example in Java

```
class BadArray {
  public static void main(String[] args) {
    int[] bad = {1, 2, 3, 4};
    System.out.printf("bad[2] = %d\n", bad[2]);
    System.out.printf("bad[4] = %d\n", bad[4]);
  }
}
```

```
WSL alinen@Xin:~/cs223/cs223-devel/chpt01$ java BadArray
bad[2] = 3
Exception in thread "main" java.lang.ArrayIndexOutOfBoundsException: Index 4
out of bounds for length 4
        at BadArray.main(BadArray.java:5)
```

# Exercise: Fibonacci Array

Write a C program the computes the first 10 Fibonacci numbers and stores them in an array.

```
WSL alinen@Xin:~/cs223/cs223-devel/chpt01$ gcc fib.c –o fib
WSL alinen@Xin:~/cs223/cs223-devel/chpt01$ ./fib
1
1
2
3
5
8
13
21
34
55
```

# Exercise: Fibonacci Array

# What about strings?

const char* message = “This is the message”;

char greeting[5] = “ciao”;

In C, a string is an **array of char**

Not every array of char is a string,
but every string is an array of char

**IMPORTANT**: strings in C MUST be large enough to hold # characters + 1

Last character is a null character that denotes the end of the string

Terminating character: `\0`

# C strings are null-terminated

- null character: \0
- char* array needs at least enough space for word + 1

# C strings

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

int main() {
  char greeting[5] = "ciao";
  const char* message = "This is the message";

  char buffer[10];
  buffer[0] = 'h';
  buffer[1] = 'i';
  buffer[2] = '\0';

  printf("%s\n", greeting);
  printf("%s\n", message);
  printf("%s\n", buffer);

  int len = strlen(message);
  printf("%d\n", len);

  strcpy(buffer, "test");
  printf("%s\n", buffer);

}
```

# string.h: helper functions for char*

To include: `#include <string.h>`

strlen

strcpy

strcat

strcmp

# Structs

No classes in C!

If we want to bundle data together in C we use a **struct**

**Classes** hold data and define methods

"know stuff" and "do stuff"

In C, only **structs** hold data and only **functions** "do stuff"

# Struct: Example

```
struct studentT {
   char name[64];
   int age;
   float gpa;
   int grad_yr;
};

 struct studentT student1, student2;

strcpy(student1.name, "Kwame Salter");  // name field is a char array
student1.age = 18 + 2;                  // age field is an int
student1.gpa = 3.5;                     // gpa field is a float
student1.grad_yr = 2020;                // grad_yr field is an int

printf("name: %s age: %d gpa: %g, year: %d\n",
        student1.name, student1.age, student1.gpa, student1.grad_yr);
```

# Aside: Using typedef for structs

```
typedef struct studentT {
   char name[64];
   int age;
   float gpa;
   int grad_yr;
} studentT;

studentT student1, student2;

strcpy(student1.name, "Kwame Salter");  // name field is a char array
student1.age = 18 + 2;                  // age field is an int
student1.gpa = 3.5;                     // gpa field is a float
student1.grad_yr = 2020;                // grad_yr field is an int

printf("name: %s age: %d gpa: %g, year: %d\n",
        student1.name, student1.age, student1.gpa, student1.grad_yr);
```

# Arrays and Structs in memory

```
struct dataT {
  int v[3];
  char c;
  float y;
};

int arr[3];
arr[0] = -1;
arr[1] = 5;
arr[2] = 3;

struct dataT data;
data.v[0] = 2;
data.v[1] = -3;
data.v[2] = 0;
data.c = ‘g’;
data.y = 3.14;
```

# Stack Diagram Practice

```
void print(const char* str, int n) {
  int a = 3;
  for (int i = 0; i < n; i++) {
    // print diagram here
    printf("%c-", str[i]);
  }
}
int main() {
  int a = 5;
  const char* greeting = "hello";
  print(greeting, a);
  return 0;
}
```

# Stack diagram practice

```
void print(const char* str,
    int n) {
    for (int i = 0; i < n; i++) {
     // print stack here
     printf("%c-", str[i]);
   }
}
int main() {
   char greeting[2];
   greeting[0] = 'a';
   greeting[1] = 'b';
   print(greeting, 2);
   return 0;
}
```