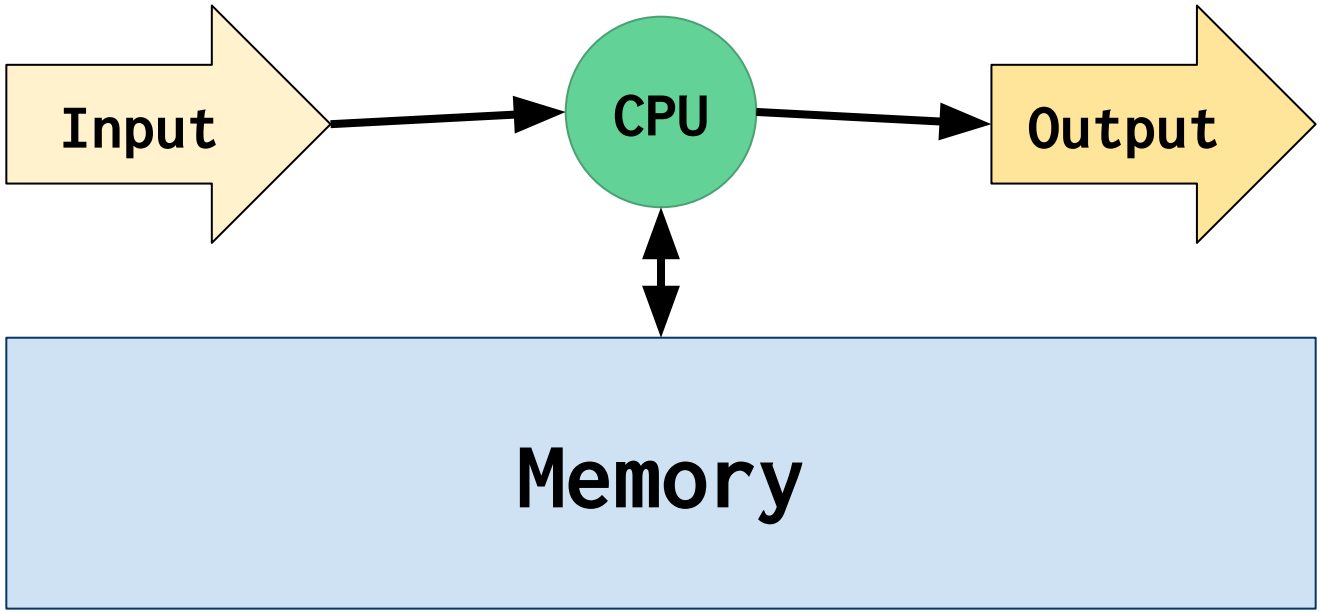
C++ Programming for FRC

Brad Thompson
Indiana FIRST Forums
10/22/2016

Brad Thompson

- Web Developer for Purdue ME
- Co-lead/programming/media mentor for FRC 461, FTC 4366

What is programming, anyway?


Input
CPU
Output
Memory

C

(1972)

C++

(1983)

Variables

```
01 int armDegrees;
02
03
04
05
06
07
08
09
10
11
12
```

Variables

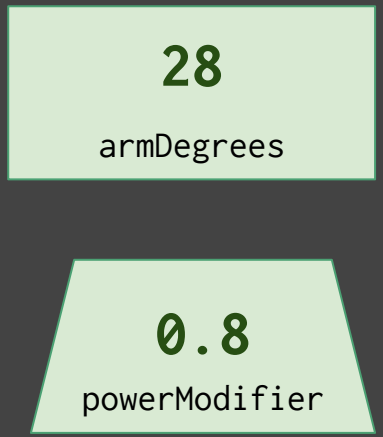
```
01 int armDegrees;
02
03
04
05
06
07
08
09
10
11
12
```

Variables

```
01 int armDegrees = 14 * 2;
02
03
04
05
06
07
08
09
10
11
12
```

Variables

```
01 int armDegrees = 14 * 2;
02
03 double powerModifier = 0.8;
04
05
06
07
08
09
10
11
12
```



Basic Data Types

int	4 byte whole number	+- 2,000,000,000
float	4 byte decimal number	+- 3.4 e +- 38
double	8 byte decimal number	+- 1.7 e +- 308
char	1 byte character	0-255
bool	1 bit	0-1
void	nothing	

Math Operators

+	Add
-	Subtract
*	Multiply
/	Divide
%	Modulus (remainder)
++	Increment (add 1)
--	Decrement (subtract 1)

&	Bitwise AND
|	Bitwise OR
^	Bitwise XOR
~	Bitwise NOT

Conditionals

```
01 if(pixyCam.foundTarget()) {
02   statusLED.set(1);
03   cout << “Target Acquired”;
04 } else {
05   statusLED.set(0);
06 }
07
08
09
10
11
12
```

Logical Operators

==	Equal to
!=	Not equal to
>	Greater than
>=	Greater than or equal to
<	Less than
<=	Less than or equal to

&&	Logical AND
||	Logical OR
!	NOT

Switch

```
switch(var) {
  case 1:
    // do stuff
    break;
  case 2:
    // do something else
    break;
  default:
    // something that will happen if nothing else happens
    break;
}

```

Loops

```
01 int count = 0;
02 while(count < 100) {
03   // do something
04   count += 1;
05 }
06
07 for(int i=0: i < 100; i++) {
08   // do something
09 }
10
11
12
```

Functions

```
01 int square(int n) {
02   return n * n;
03 }
04
05 int fiveSquared = square(5);
06
07
08
09
10
11
12
```

Functions

```
01 int calculateArmMovement(int current, int setpoint, int last) {
02   // complicated math-y stuff
03   return x;
04 }
05
06 armMotor.setPower(calculateArmMovement(a, b, c));
07
08
09
10
11
12
```

Functions

```
01 void calculateArmMovement(int current, int setpoint, int last) {
02   // complicated math-y stuff
03   armMotor.setPower(x);
04 }
05
06 calculateArmMovement(a, b, c);
07
08
09
10
11
12
```

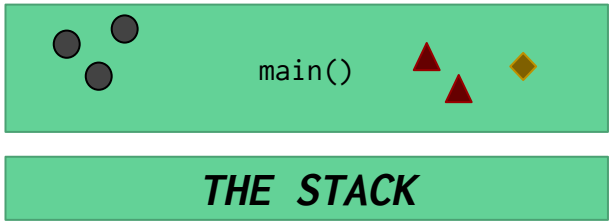
Classes and Objects

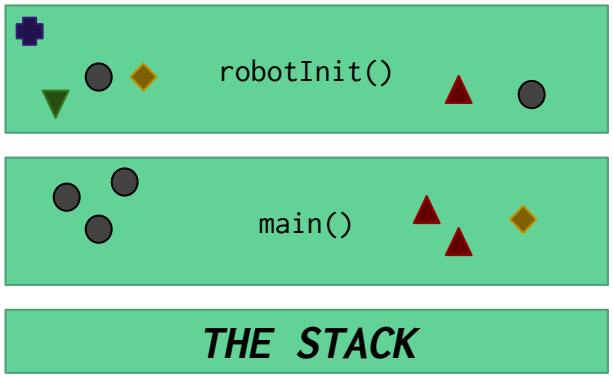
```
01 class Robot {
02   public:
03     double leftSpeed;
04     double rightSpeed;
05     bool enabled;
06 }
07
08 Robot myRobot;
09 myRobot.leftSpeed = 0.6;
10 myRobot.enabled = true;
11
12
```

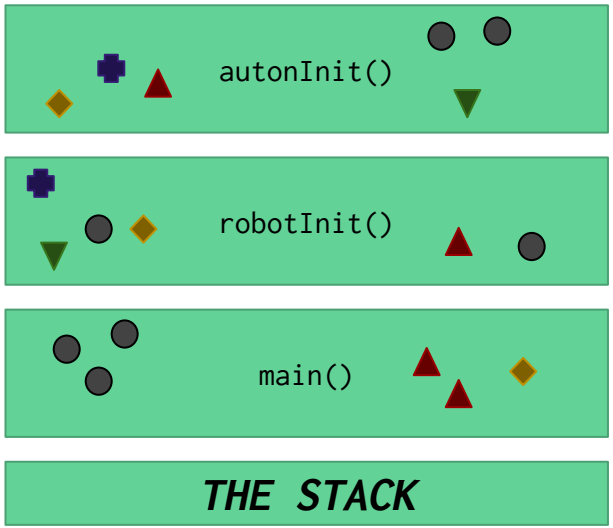
Stacks, Heaps, References, and Pointers

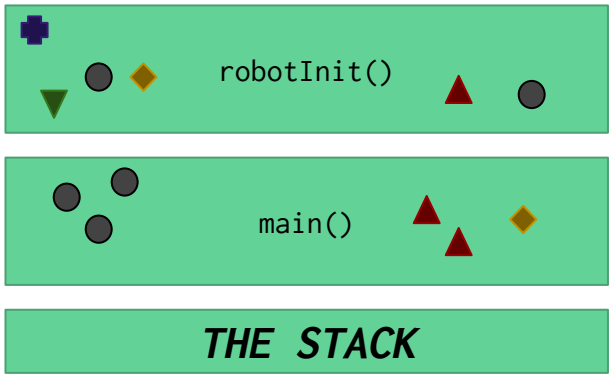
Memory Areas

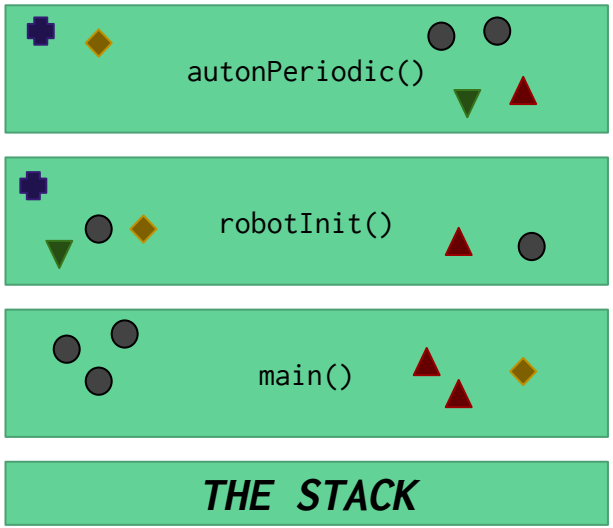
- **Program**: the actual code you're running
- **Data**: globally defined variables with pre-defined data
- **BSS**: globally defined variables that haven't been defined
- **Heap**: a big pile of dynamically assignable memory
- **Stack**: keeps track of the locally defined variables in functions

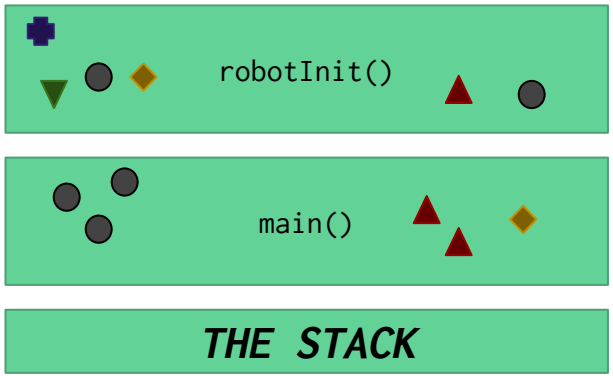

main()
THE STACK

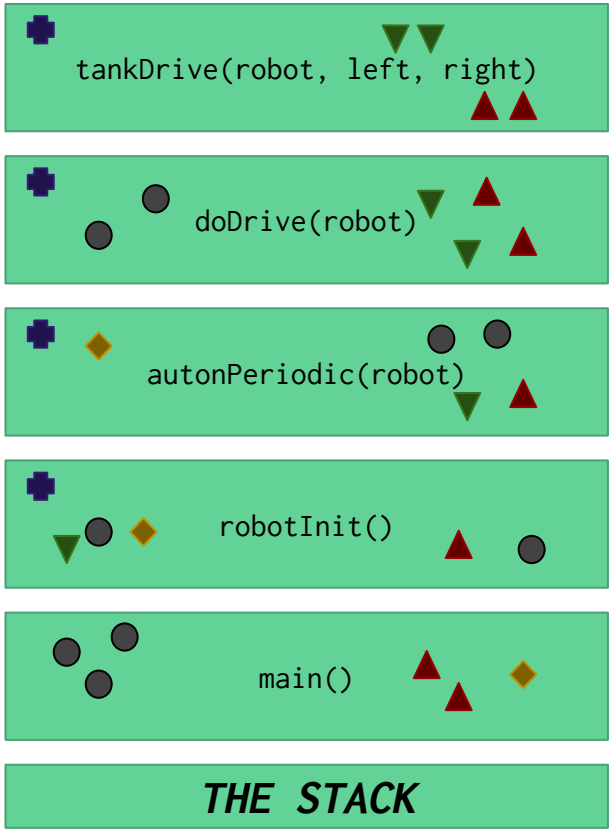

robotInit()
main()
THE STACK


autonInit()
robotInit()
main()
THE STACK

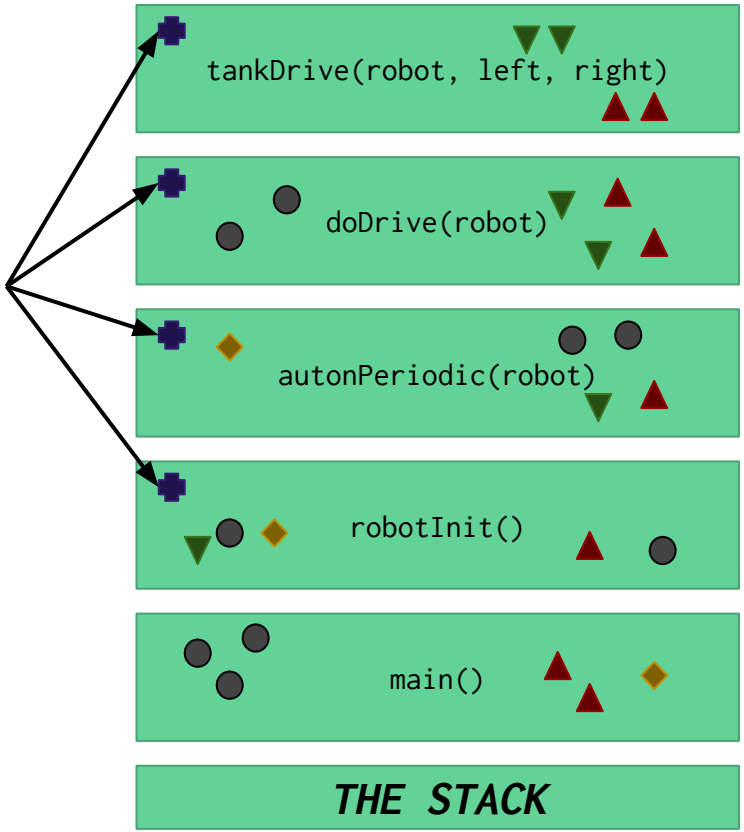

robotInit()
main()
THE STACK


autonPeriodic()
robotInit()
main()
THE STACK

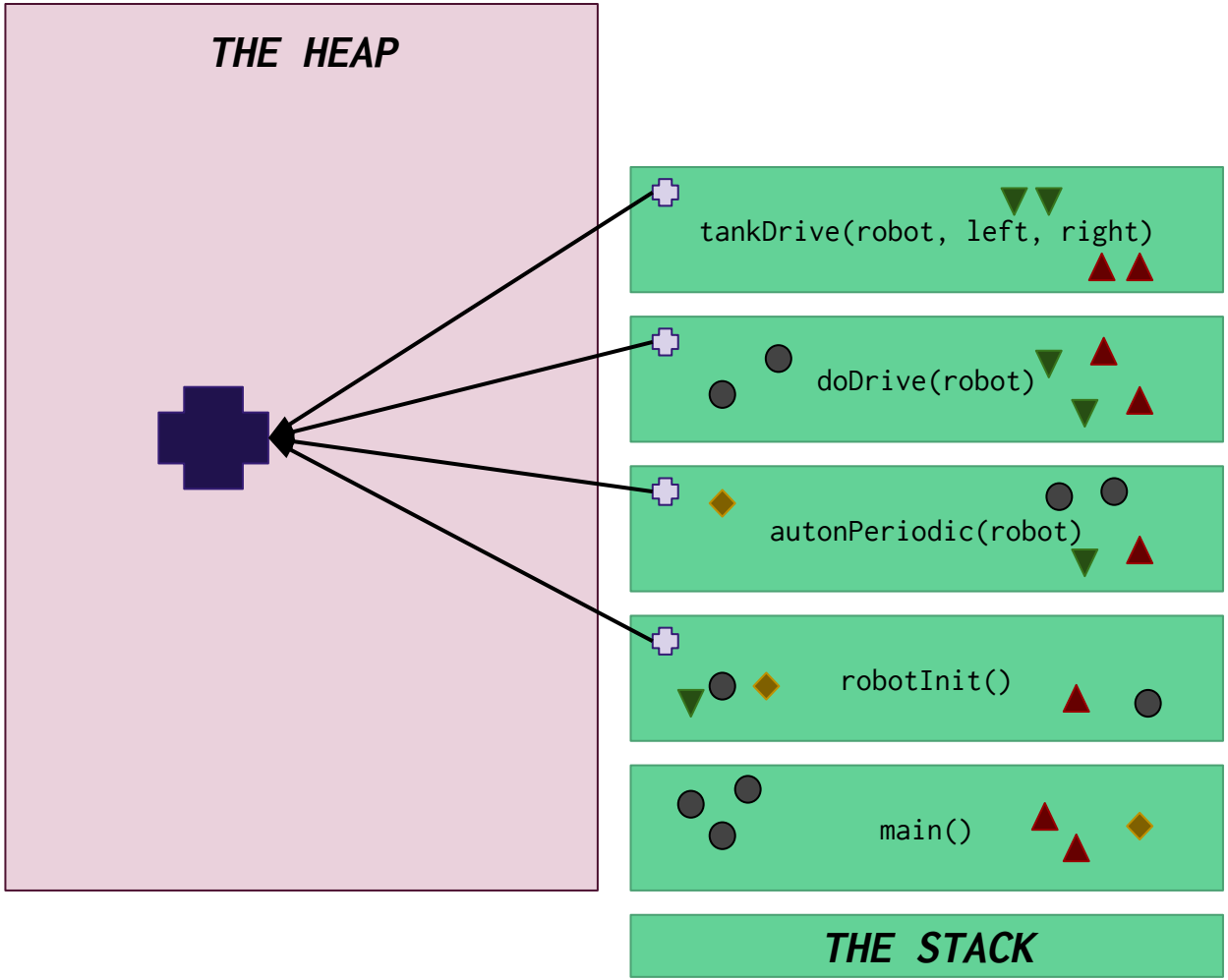

robotInit()
main()
THE STACK


tankDrive(robot, left, right)
doDrive(robot)
autonPeriodic(robot)
robotInit()
main()
THE STACK

4 different copies
of our object!



Instead of making copies of objects, let’s just make one object & pass around the location in memory of that object.


THE HEAP
tankDrive(robot, left, right)
doDrive(robot)
autonPeriodic(robot)
robotInit()
main()
THE STACK

Pointers & Addresses

```
01 RobotDrive myRobot;  // this is an actual RobotDrive
02
03 RobotDrive *myRobotPointer; // stores the address of a RobotDrive
04
05 myRobotPointer = &myRobot; //myRobotPointer now points to myRobot
06
07 void robotStatus(RobotDrive *r) { ... }
08
09 RobotDrive *buildRobot() { ... }
10
11
12
```

Pointers can be kind of tricky & easy to mess up. C++ also has references, which are basically permanent, never-empty pointers.

C++ References

```
01 RobotDrive myRobot;  // this is an actual RobotDrive
02
03 RobotDrive &myRobotRef = myRobot ; // stores the address of a
04                                    // RobotDrive
05
06 void robotStatus(RobotDrive& r) { ... }
07
08 RobotDrive& buildRobot() { ... }
09
10
11
12
```

Pointer & Object Operators

*	Follow/create pointer
&	Address of / create reference
.	Object variable/method
->	Follow pointer/reference to variable/method
::	Access class/namespace variables

WPILib

WPILib

- WPILib is a big collection of classes and resources to help write FRC robot code.
- Defines classes for motor controllers, joysticks, sensors, ...
- Also ships with sample code

WPILib

- A new version is released at kickoff every year.
- Changes coming in 2017:
  - Component-specific parts of the code are being pulled out of the core library & implemented as independent libraries.

WPILib

- WPILib ships with 2 base robots:
  - IterativeRobot follows a more “traditional” setup & control loop model
  - Command-based Robot is a more object-oriented, event-driven model
    - Use the **Robot Builder** utility to set up your robot structure, inputs, mechanisms, and commands

WPILib Motor Controllers

```
01 Talon *armMotor = new Talon(6); //controls a Talon on PWM 6
02
03 armMotor->Set(0.7);  //takes value from -1.0 to 1.0
04
05
06
07
08
09
10
11
12
```

WPILib Drivetrain

```
01 RobotDrive myRobot(0, 1); //initialize 2-motor drivetrain
02                           //defaults to Talons
03
04 RobotDrive myJagRobot(leftDrive, rightDrive); //or pass in motor
05                                               //controller objects
06
07 RobotDrive myFourDrive(frontLeft, backLeft, frontRight, backRight);
08
09
10
11
12
```

WPILib Driving

```
01 myRobot.ArcadeDrive(stick);  //pass in a 2 axis joystick…
02 myRobot.ArcadeDrive(moveAxis, rotateAxis); //manually specify axes…
03 myRobot.ArcadeDrive(move, rotate); //or pass in raw values
04
05 myRobot.TankDrive(leftAxis, rightAxis); //pass in 2 axes…
06 myRobot.TankDrive(leftSpeed, rightSpeed); //or raw values
07
08
09
10
11
12
```

WPILib Limit Switches

```
01 DigitalInput *limitSwitch  = new DigitalInput(1);
02
03 if(limitSwitch->Get()) {
04   cout << “Yay!”;
05 }
06
07
08
09
10
11
12
```

WPILib Ultrasonic Distance Sensor

```
01 Ultrasonic *distance = new Ultrasonic(1,2);
02 distance->SetAutomaticMode(true);
03
04 cout << distance->GetRangeInches();
05
06
07
08
09
10
11
12
```

WPILib Encoder

```
01 Encoder *enc = new Encoder(3, 4, false, Encoder::EncodingType::k4X);
02 enc->Reset();
03
04 enc->Get();
05
06
07
08
09
10
11
12
```

SmartDashboard

- SmartDashboard is a piece of software that runs with the driver station to get information to & from the robot.
- It works like a dictionary - you associate a “key” with a “value”
- If you go command-based, you can start commands directly from the dashboard.

SmartDashboard

```
01 SmartDashboard::PutNumber(‘key’, 7);
02
03 SendableChooser chooser;
04 chooser.AddObject(‘Default’, 1)
05 chooser.AddObject(‘Not Default’, 2);
06 // somewhere later…
07 choser.GetSelected();
08
09
10
11
12
```

Other Output

- You can use the usual console output functions to write text back to the drivers station console & the RioLog in Eclipse.
- You can also create files on the RoboRio to log & store data.

Questions?

https://wpilib.screenstepslive.com/s/4485

http://first.wpi.edu/FRC/roborio/release/docs/cpp/

bkt@brad-thompson.com