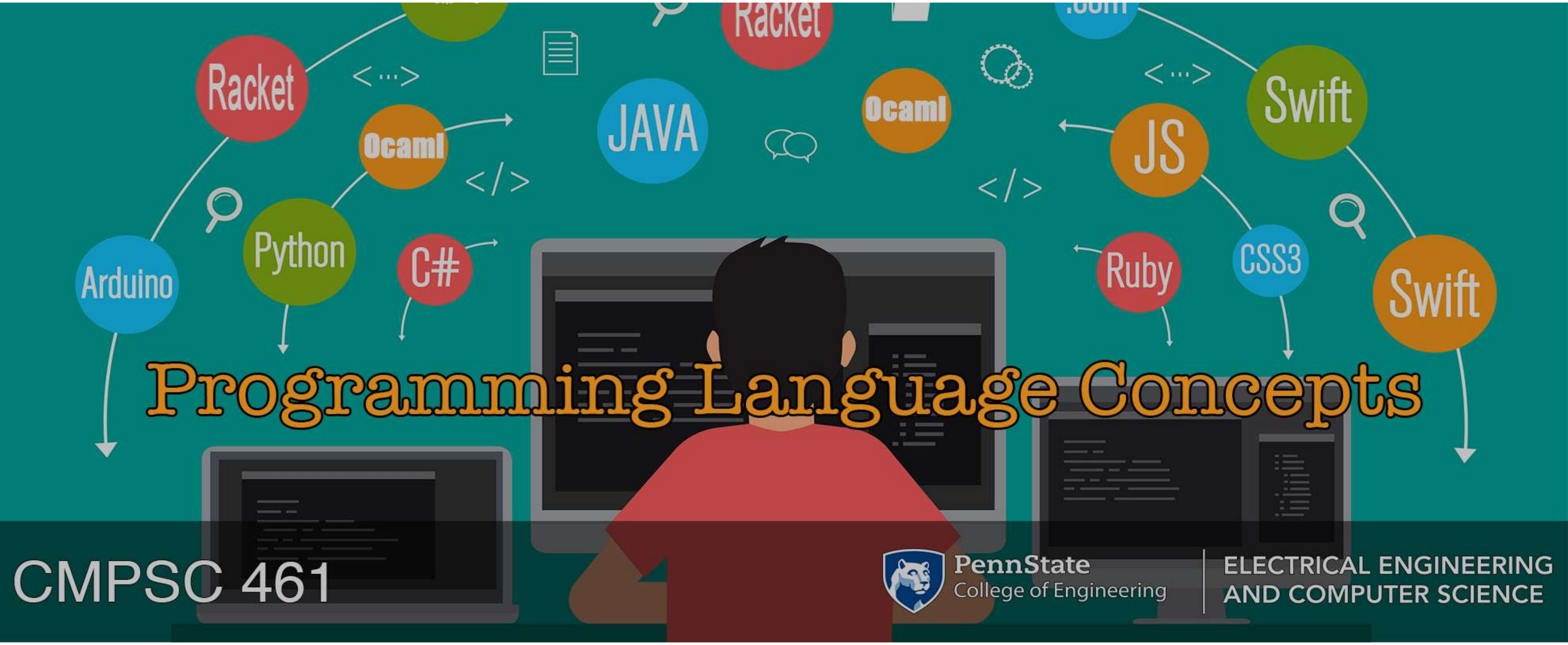

Program Verification

Professor: Suman Saha

Correctness

A program implements the desired property for all possible inputs

E.g.,

Functional correctness: a program calculates n!

Type safety: no typing errors at run time

Memory safety: no buffer overflow in a program

Security: no information leakage, no violation of integrity, …

We will focus on **functional correctness** in this course

Functional Correctness

```
r:=0, i:=0;
while (i<n) {
  r := r+2;
  i ++;
}
```

This code ensures $r = 2\times n$ when $n \geq 0$?

- ***Testing***: assert `r:= 2*n` and execute with different values of n (cannot cover all inputs in general)
- ***Verification***: prove $r = 2\times n$ for any possible n

Program and Logics

Is a program correct (e.g., is the result n!)?

We need to formally specify

1) The desired property
2) The behavior of program

Logics as our specification language

Program and Logics

We need to formally specify

1) The desired property

```
int Max(int a, int b) {
  int m;
  if (a>b)  m:=a;
  else      m:=b;
  return m;
}
```

Assignment

Precondition (true)

function symbol in logics

Postcondition $(m = \max(a, b))$

Program and Logics

We need to formally specify

1) The desired property

```
int factorial(int n) {
  int r:=1, i=n;
  while (i>0) {
    r := r*i;
    i --;
  }
  return r;
}
```

Precondition $(\mathrm{n} \geq 0)$

Postcondition $(r = n!)$

Program and Logics

Is a program correct?

We need to formally specify

1) The desired property
2) **The behavior of program**

Informally....

```
x := 5;
y := 1;
```

After second assignment, we know $\{x = 5, y = 1\}$

Why?

1. Initially, we assume nothing
2. After the first assignment, we know $\{x = 5\}$
3. After the second assignment, we know $\{y = 1\}$ is true as well

Formalizing the Reasoning

```
x := 5;
y := 1;
```

1. Initially, we assume nothing
2. After the first assignment, we know $\{x = 5\}$
3. After the second assignment, we know $\{y = 1\}$ is true as well

The reasoning:

$$\{\text{true}\}\texttt{x:=5}\ \{x = 5\}\ \texttt{y:=1}\ \{x = 5 \wedge y = 1\}$$

Each predicate specifies the assertion that must be true before/after a statement

Hoare Triple

Assertion: a predicate that describes the ***state*** of a program at a point in its execution

Hoare Triple: $\{P\}s\{Q\}$

Precondition P: an assertion before execution

Postcondition Q: an assertion after execution

Program s: program being analyzed

A triple is ***valid*** If we start from a state satisfying P, and execute s, then final state must satisfy Q

Examples

$\{\text{true}\}\texttt{x:=5}\{x = 5\}$

$\{y = 6\}\texttt{x:=5}\{x = 5, y = 6\}$

$\{\text{true}\}\texttt{x:=5}\{x < 10\}$

$\{x = y\}\texttt{x:=x+3}\{x = y + 3\}$

$\{x = a\}\texttt{if (x<0) then x:=-x}\ \{x = |a|\}$

All of these triples are valid

Overview of Program Verification

```
x := 5;
y := 1;
```

x+y>5 after execution from any initial state?

Write down specification

{true}

```
x := 5;
y := 1;
```

{x+y>5}

known as Hoare triple

Check if the Triple is valid

Program is verified if the triple is valid

A triple $\{P\}s\{Q\}$ is ***valid*** If we start from a state satisfying P, and execute s, then final state must satisfy Q

Program Verification

Goal: check if $\{P\}s\{Q\}$ is valid

$\{\text{true}\}\texttt{x:=5}\{x = 5\}$

$\{\text{true}\}\texttt{x:=5}\{x = 5 \lor x = 2\}$

$\{\text{true}\}\texttt{x:=5}\{x > 0\}$

$\{\text{true}\}\texttt{x:=5}\{x < 10\}$

Observation: some postconditions are more useful

$x = 5 \Rightarrow x = 5 \lor x = 2$

$x = 5 \Rightarrow x > 0$

$x = 5 \Rightarrow x < 10$

Need to compute the ***strongest*** postcondition

Program Verification

Compute strongest postcondition and check the truth value

$$sp(x := 5; y := 1, true) \Rightarrow x + y > 5?$$

```
x := 5;
y := 1;
```

x+y>5 after execution from any initial state?

Write down specification

{true}

```
x := 5;
y := 1;
```

{x+y>5}

Check if the Triple is valid

Program is verified if the triple is valid

Program Verification

Goal: check if $\{P\}s\{Q\}$ is valid
Method 1: check $\text{sp}(s, P) \Rightarrow Q$
Method 2: check $P \Rightarrow \text{wp}(s, Q)$

Weakest Precondition

$\text{wp}(s, Q)$ is the **weakest precondition** of s, w.r.t. Q
Property: $\{P\}s\{Q\}$ is valid iff $P \Rightarrow \text{wp}(s, Q)$

Hence, validity of a triple $\{P\}s\{Q\}$ is equivalent to the truth value of proposition $P \Rightarrow \text{wp}(s, Q)$

Assignment Rule (Hoare's Axiom)

$$\text{wp}(\mathrm{x} := \mathrm{e}, Q) = Q[x \leftarrow e]$$

Examples:

$\text{wp}(\mathrm{x} := 5, x = 5) = (5 = 5) = (\text{true})$

$\text{wp}(\mathrm{x} := \mathrm{x} + 3, x = y + 3) = (x + 3 = y + 3)$

$= (x = y)$

This rule is simpler than Floyd's axiom, hence weakest precondition is used in most systems

Composition Rule

$$wp(\mathtt{s1;s2}, Q) = wp(\mathtt{s1}, wp(\mathtt{s2}, Q))$$

```
{true}
x := 5;
y := 1;
```
$\{(y=1) \wedge (x=5)\}$

$$
\begin{aligned}
& wp(\mathtt{x:=5;y:=1}, (x = 5) \wedge (y = 1)) \\
&= wp(\mathtt{x:=5}, wp(\mathtt{y:=1}, (x = 5) \wedge (y = 1))) \\
&= wp(\mathtt{x:=5}, (x = 5) \wedge (1 = 1)) \\
&= (5 = 5) \wedge (1 = 1) \\
&= \text{true}
\end{aligned}
$$

Composition Rule

$$\text{wp}(\texttt{s1;s2}, Q) = \text{wp}(\texttt{s1}, \text{wp}(\texttt{s2}, Q))$$

```
{true}
x := 5;
x := 2;
{x=2}
```

$$\begin{aligned} & \text{wp}(\texttt{x:=5;x:=2}, x = 2) \\ = & \text{ wp}(\texttt{x:=5}, \text{wp}(\texttt{x:=2}, x = 2)) \\ = & \text{ wp}(\texttt{x:=5}, 2 = 2) \\ = & \ (2 = 2) \\ = & \text{ true} \end{aligned}$$

Branch Rule

$$\boxed{\begin{aligned}&\text{wp}(\text{if}(E)\ \texttt{s1}\ \text{else}\ \texttt{s2}, Q) = \\ &\quad (E \Rightarrow \text{wp}(\texttt{s1}, Q) \wedge \neg E \Rightarrow \text{wp}(\texttt{s2}, Q))\end{aligned}}$$

```
{true}
if (x>0)
  y := x;
else
  y := -x;
{y≥0}
```

$$\begin{aligned}&\text{wp}(\text{P}, y \geq 0)\\ &= x > 0 \Rightarrow \text{wp}(\text{y} := \text{x}, y \geq 0) \wedge\\ &\neg(x > 0) \Rightarrow \text{wp}(\text{y} := -\text{x}, y \geq 0))\\ &= (x > 0 \Rightarrow x \geq 0) \wedge (x \leq 0 \Rightarrow -x \geq 0)\\ &= \text{true}\end{aligned}$$

Computing WP

Goal: check if $\{P\}s\{Q\}$ is valid
Method 1: check sp$(s, P) \Rightarrow Q$
Method 2: check $P \Rightarrow$wp(s, Q)

$$\text{wp}(x := e, Q) = Q[x \leftarrow e]$$
$$\text{wp}(s_1;\ s_2, Q) = \text{wp}(s_1, \text{wp}(s_2, Q))$$
$$\text{wp}(\text{if}(E)\ s_1 \text{ else } s_2, Q) =$$
$$(E \Rightarrow \text{wp}(s_1, Q) \wedge \neg E \Rightarrow \text{wp}(s_2, Q))$$
$$\text{wp}(\text{nop}, Q) = Q$$

A dummy operation that has no effects

Example

```
{x>0}
x := x+1;
y := x * (x+5);
{y>0}
```

$\text{wp}(x := e, Q) = Q[x \leftarrow e]$
$\text{wp}(s_1;\ s_2, Q) = \text{wp}(s_1, \text{wp}(s_2, Q))$
$\text{wp}(\text{if}(E)\ s_1 \text{else}\ s_2, Q) =$
$\quad (E \Rightarrow \text{wp}(s_1, Q) \land \neg E \Rightarrow \text{wp}(s_2, Q))$
$\text{wp}(\text{nop}, Q) = Q$

Goal: show the Hoare triple is valid

1) Compute wp(prog, postcondition)

2) Show the precondition implies wp

Program Verification

Loops $\{P\}\text{while } (E)\ s\ \{Q\}$

What is the WP?

Let W= $\text{while } (E)\ s$, then $\{P\}\text{while } (E)\ s\ \{Q\}$

is the same as $\{P\}\text{if } (E)\ s; \text{W else nop } \{Q\}$

By if-rule,

$$\begin{aligned} \text{wp}(W, Q) &= (E \Rightarrow \text{wp}(s; W, Q) \wedge \neg E \Rightarrow Q) \\ &= (E \Rightarrow \text{wp}(s; \text{wp}(W, Q)) \wedge \neg E \Rightarrow Q) \end{aligned}$$

Loop Invariant

Program Verification

Loop Invariant $\{P\}\text{while } (E)\ s\ \{Q\}$

$$Inv \Rightarrow (E \Rightarrow \text{wp}(s, Inv) \wedge \neg E \Rightarrow Q)$$

Hence, $Inv \wedge E \Rightarrow \text{wp}(s, Inv)$ and $Inv \wedge \neg E \Rightarrow Q$

(Proof is beyond the scope of this lecture)

Loop invariant (Inv) is a proposition that is:

1) Initially true ($P \Rightarrow Inv$)
2) True after each iteration ($Inv \wedge E \Rightarrow \text{wp}(s, Inv)$)
3) Termination of loop implies the postcondition ($Inv \wedge \neg E \Rightarrow Q$)

Loop Invariant and Induction

Loop invariant (Inv) is a proposition that is:

1) Initially true ($P \Rightarrow Inv$)
2) True after each iteration ($Inv \land E \Rightarrow \mathrm{wp}(s, Inv)$)
3) Termination of loop implies the postcondition ($Inv \land \neg E \Rightarrow Q$)

Intuitively, we are proving the correctness of an arbitrary number of loop iterations, by **induction**!

Program Verification

Example

```
{n ≥ 0}
r:=0, i:=0;
while (i<n) {
  r := r+2;
  i ++;
}
{r = 2×n}
```

Goal: show the Hoare triple is valid

1) Write down a tentative loop invariant (Inv)

   $r = 2 \times i \land i \leq n$

2) Show *Inv* is a loop invariant

- $\{n \geq 0\}$ r:=0, i=0; $\{Inv\}$ is valid
- $Inv \land i < n \Rightarrow$ wp(r:=r+2; i++, Inv)
- $Inv \land i \geq n \Rightarrow r = 2 \times n$

Total vs. Partial Correctness

$\{P\}\text{while }(E)\ s\ \{Q\}$

Partial correctness: if the loop terminates, Q must be true. *However, the loop might not terminate*

E.g., $\{P\}\text{while (true) } s\ \{Q\}, Inv \wedge \neg\text{true} \Rightarrow Q$ is true

The loop invariant only enforces partial correctness

Total correctness: prove loop determinates (undecidable in general)

Summary

Goal: prove a program s is correct

Step 1: formalize "correctness" by writing down the precondition P and postcondition Q

Step 2: show that the Hoare tripe ($\{P\}s\{Q\}$) is valid

- Mostly automatic, except for the loops

What is verified?

Given any state satisfying *P*, the final state after executing s must satisfy Q, if s terminates