As you arrive:

1. Start up your computer and plug it in
2. ***Log into Angel*** and go to CSSE 120
3. Do the ***Attendance Widget*** – the PIN is on the board
4. Go to the course ***Schedule Page***
5. Open the ***Slides*** for today if you wish
6. Check out today's project: `25-CPointers`

Plus in-class time working on these concepts AND practicing previous concepts, continued as homework.

Pointers

- What they are. Why they are useful.
- Their notation in C: & * *
- Using pointers to get data back from a function. scanf as example.
- Next time: Using pointers to send a reference to lots of data to a function

Session 25

CSSE 120 – Introduction to Software Development

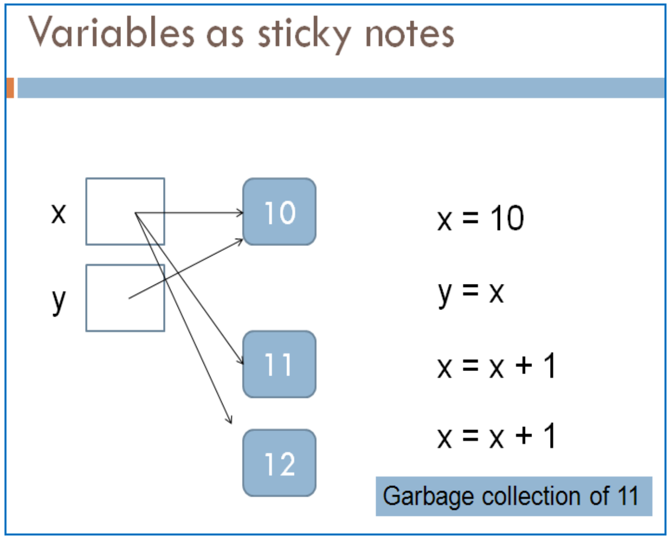
Outline

- Previously: C basics
  - Functions and variables, with types
  - FOR and WHILE loops
  - IF statements
  - Input, via *scanf*
- Structures
  - What they are
  - How to use them
  - Header files

- Today: Pointers
  - What they are.
    Why they are useful.
  - Their notation in C
    & * *
  - Pointers vs Pointee's – deferencing
  - Using pointers to:
    - Mutate variables in the calling function
    - Get data back from a function
      - *scanf* as an example
    - Send a reference to lots of data to a function (arrays – next time)

Variables and parameter passing in Python

- Recall that in Python "everything is an object" and hence all variable names are ***references*** to objects
  - They act like sticky notes →
  - When we pass a variable to a function, we are passing a reference to an object.
    - This is efficient (fast) – we copy only the reference, not all the data that is referenced. For example, when we pass a list, we pass a reference to the list, not all the data in the list.
    - If the object is mutable, we can mutate it in the function – this is convenient and efficient. If the object is not mutable, we are assured that it is unchanged when we return from the function – this makes it easier to write correct code. So both mutable and immutable objects have their place.

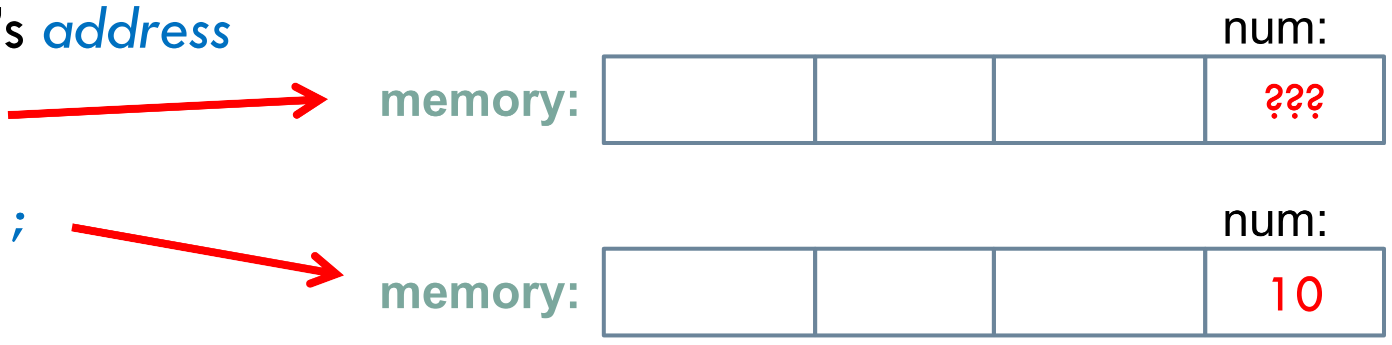


Q1

Variables in C

- Variables are stored in memory
  - We call the place in memory the variable's *address*

```
int num;
num = 10;
```



- C has several types of variables:
  - Integers – their bits are interpreted as a whole number
  - Doubles – their bits are interpreted as a floating point number
  - …
  - ***Pointers*** – their bits are interpreted as an ***address in memory***
    - As such, they are *references* to other data

Q2-4

The three notations for pointers in C

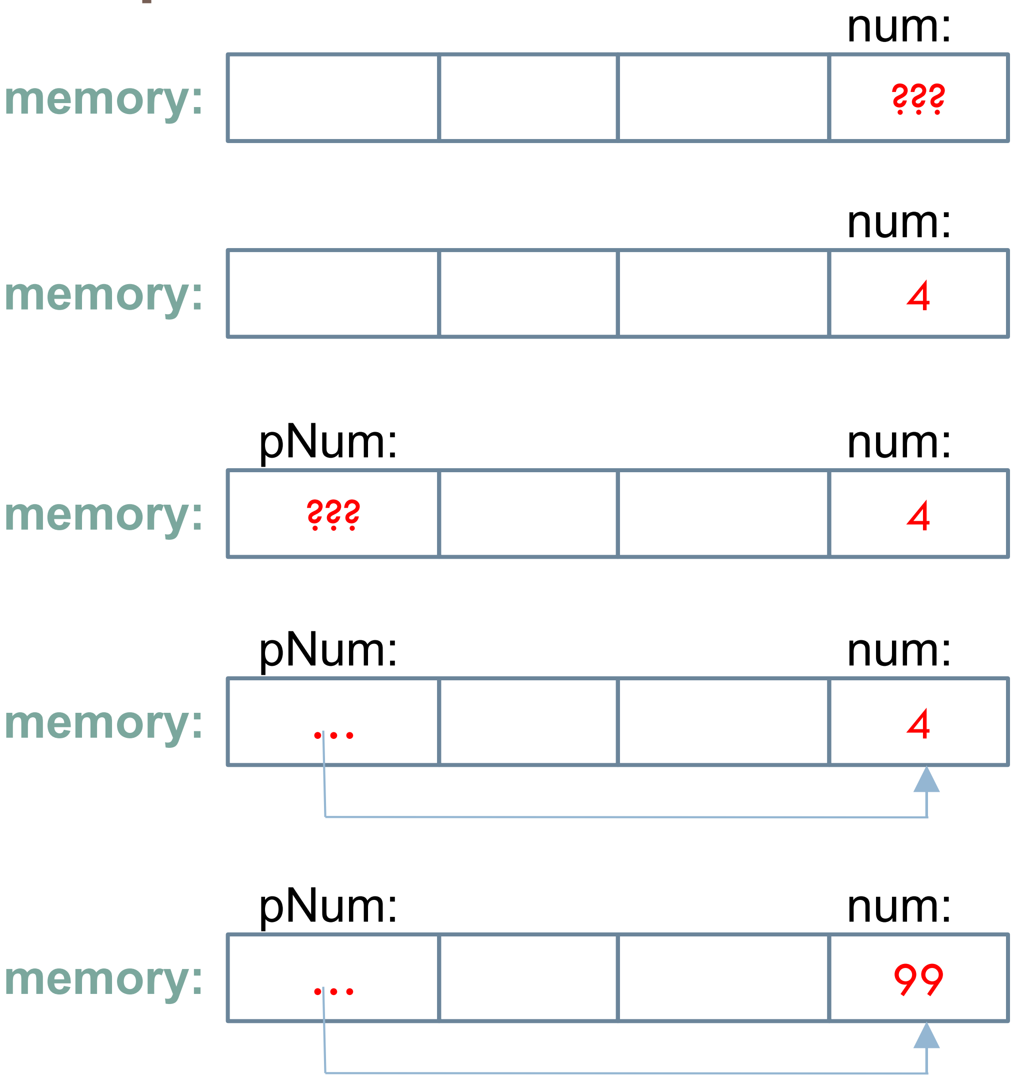
```
int num;

num = 4;

int *pNum;

pNum = &num;

*pNum = 99;
```



pNum is a ***pointer*** to an **int**

pNum is set to the ***address*** of **num**

The ***thing at pNum*** is set to **99**

pNum is the *pointer* and **num** is the *pointee*.
***pNum** *deferences* the pointer, which means that it obtains the pointee.

Q5-8

Here's Binky!

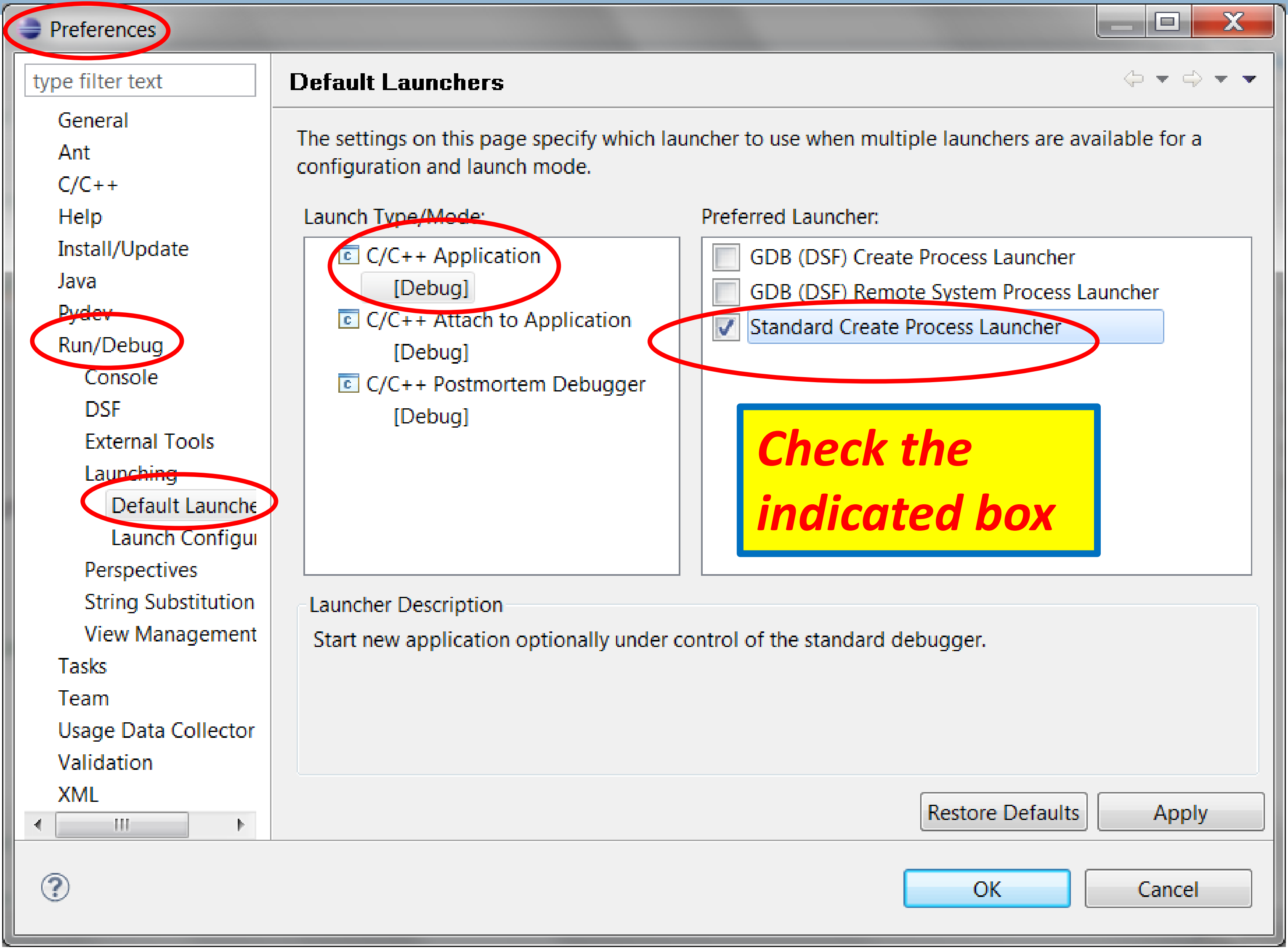
- Ignore *malloc* in the video for now
- Vocabulary
  - *Pointee*: the thing referenced by a pointer
  - *Dereference*: obtain the pointee
- See http://cslibrary.stanford.edu/104/
- What name did we give pointer "sharing" in Python?
  - Answer: *aliasing*

Checkout today's exercise: `Session25-Cpointers`
Then configure your debugger:

From:

Window ~ Preferences

Continues on the next slide



Preferences

type filter text

- General
- Ant
- C/C++
- Help
- Install/Update
- Java
- Pydev
- Run/Debug
  - Console
  - DSF
  - External Tools
  - Launching
    - Default Launche
    - Launch Configu
  - Perspectives
  - String Substitution
  - View Management
- Tasks
- Team
- Usage Data Collector
- Validation
- XML

Default Launchers

The settings on this page specify which launcher to use when multiple launchers are available for a configuration and launch mode.

Launch Type/Mode:

- C/C++ Application
  - [Debug]
- C/C++ Attach to Application
  - [Debug]
- C/C++ Postmortem Debugger
  - [Debug]

Preferred Launcher:

- [ ] GDB (DSF) Create Process Launcher
- [ ] GDB (DSF) Remote System Process Launcher
- [✓] Standard Create Process Launcher

Check the indicated box

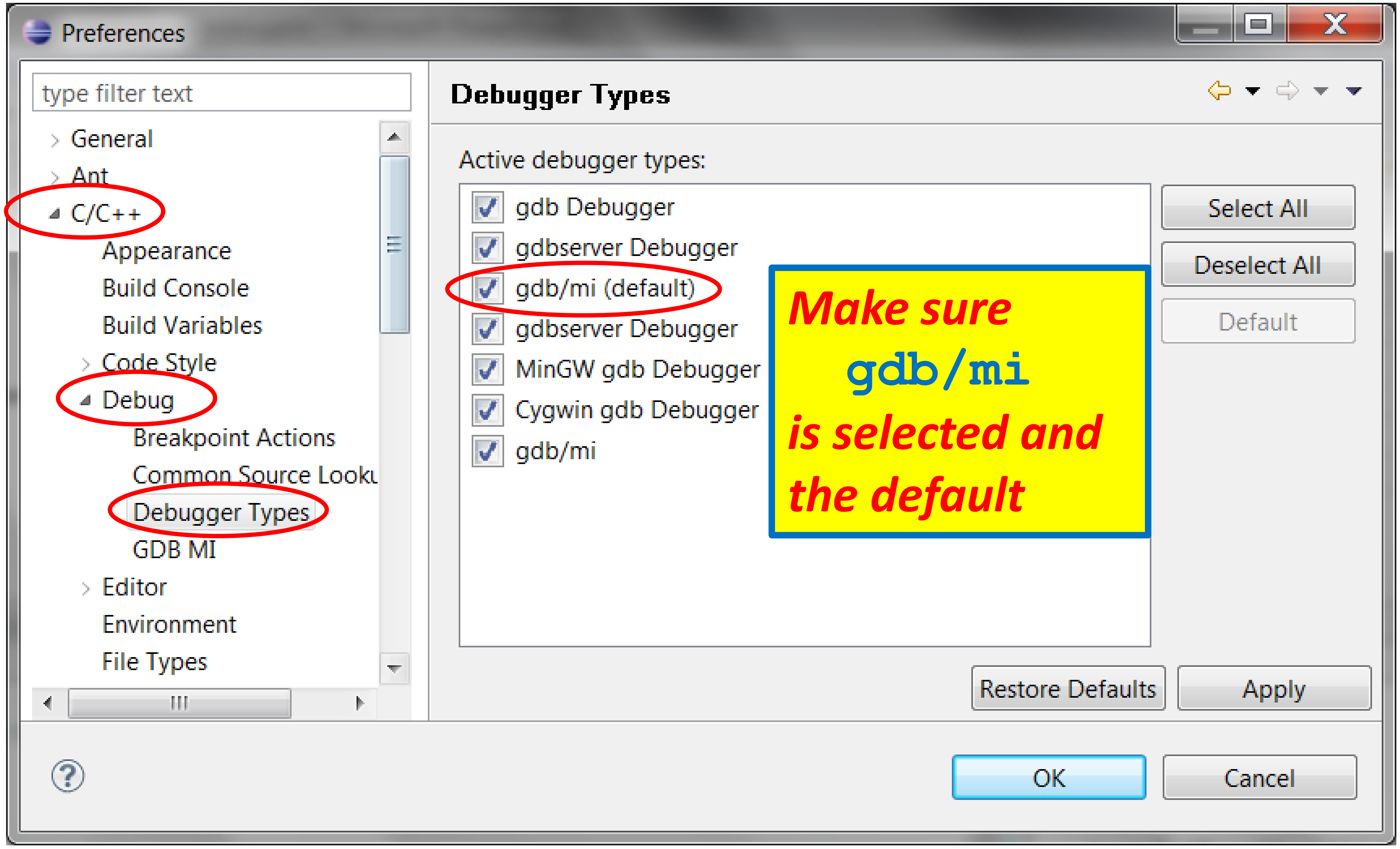
Launcher Description

Start new application optionally under control of the standard debugger.

Continue to configure your debugger

From: **Window ~ Preferences**

Continues on the next slide


Preferences

type filter text

- General
- Ant
- C/C++
  - Appearance
  - Build Console
  - Build Variables
  - Code Style
  - Debug
    - Breakpoint Actions
    - Common Source Looku
    - Debugger Types
    - GDB MI
  - Editor
  - Environment
  - File Types

Debugger Types

Active debugger types:

- [✓] gdb Debugger
- [✓] gdbserver Debugger
- [✓] gdb/mi (default)
- [✓] gdbserver Debugger
- [✓] MinGW gdb Debugger
- [✓] Cygwin gdb Debugger
- [✓] gdb/mi

Make sure `gdb/mi` *is selected and the default*

Continue to configure your debugger

Do:

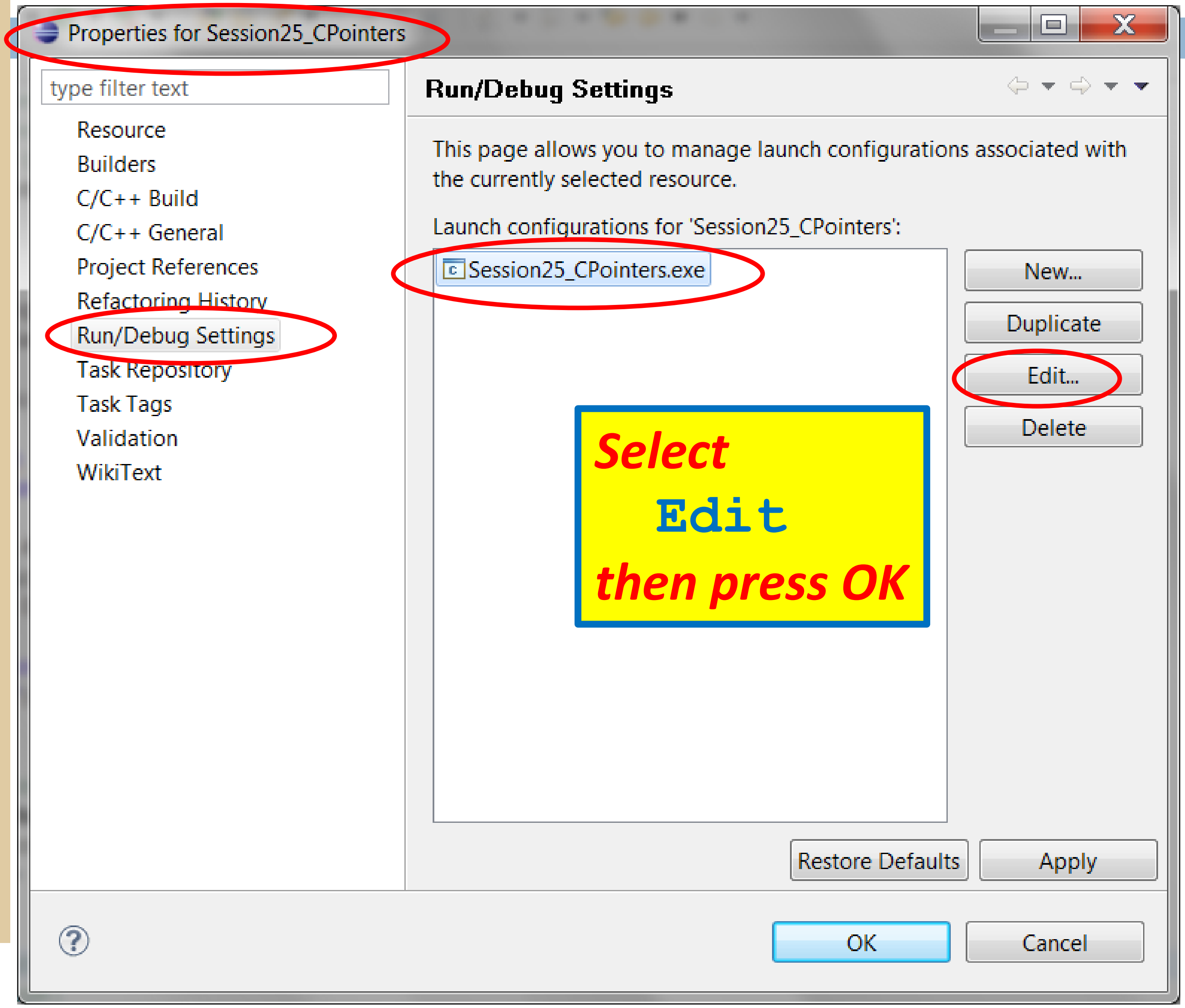
Project ~ Clean

Then:

Right-click on **Session25_ Cpointers** in **Project Explorer**

and select **Properties**

Continues on the next slide



Properties for Session25_CPointers

type filter text

- Resource
- Builders
- C/C++ Build
- C/C++ General
- Project References
- Refactoring History
- Run/Debug Settings
- Task Repository
- Task Tags
- Validation
- WikiText

Run/Debug Settings

This page allows you to manage launch configurations associated with the currently selected resource.

Launch configurations for 'Session25_CPointers':

Session25_CPointers.exe

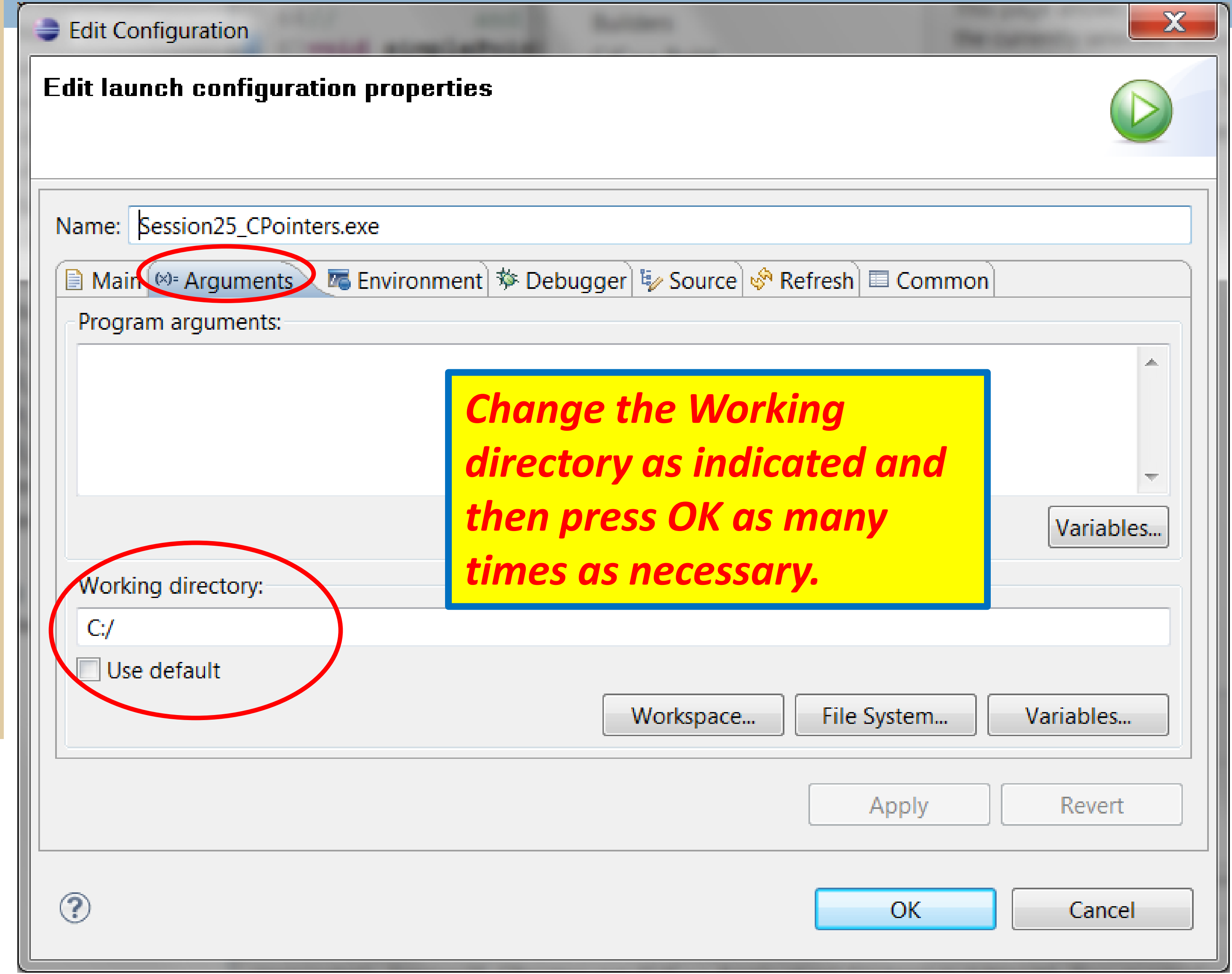
Select* `Edit` *then press OK

Continue to configure your debugger

Select the ***Arguments*** tab.

Uncheck ***Use default*** and change the Working Directory to: C : /

Continues on the next slide



Edit Configuration

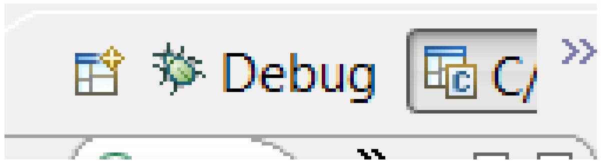
Edit launch configuration properties

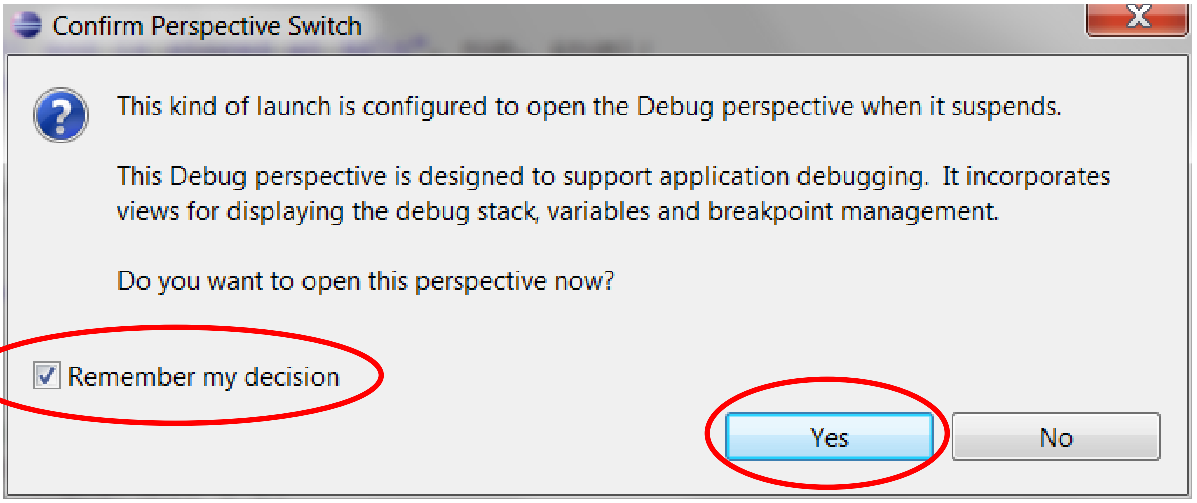
Name: Session25_CPointers.exe

Change the Working directory as indicated and then press OK as many times as necessary.

Run your debugger

- Run the program in the debugger.
  - You may get error messages of the form "*No such file or directory*" but as long as you get into the Debug mode, no problem.
  - If you see the dialog shown below, CHECK THE BOX and select Yes.
- Single-step through the program in the debugger to confirm that all is OK
- Switch back and forth between the Debug and C/C++ perspectives
- Whenever you leave the Debug perspective, be sure to stop the run




Confirm Perspective Switch

This kind of launch is configured to open the Debug perspective when it suspends.

This Debug perspective is designed to support application debugging. It incorporates views for displaying the debug stack, variables and breakpoint management.

Do you want to open this perspective now?

☑ Remember my decision

[✓ Yes] No

Proof that pointers store addresses

- Checkout today's exercise:
  `Session25-CPointers`
- Do ***TODO 1*** and ***TODO 2***. As part of TODO 2, answer the quiz questions.
  - When instructed to do so, run it in the debugger
    - Use the Debug view
    - It automatically inserts a breakpoint at the start of `main`
    - Single-step from there to answer the questions
    - You may get error messages of the form "*No such file or directory*" but as long as you get into the Debug mode, no problem.

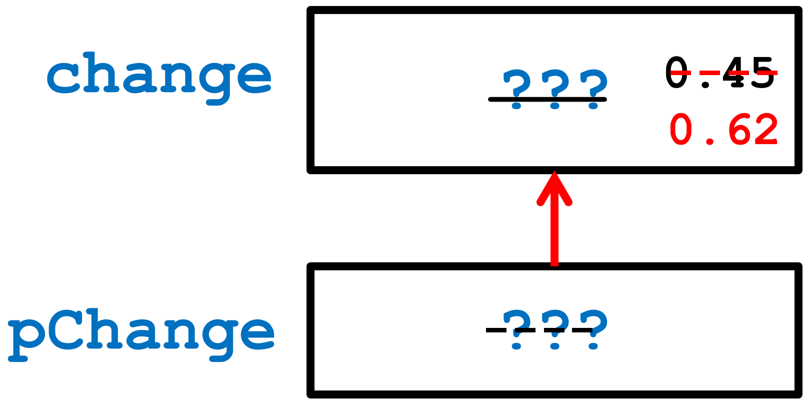
Q9-11

Box and pointer diagrams

- Together, let's draw a Box-and-Pointer diagram for some of the variables in ***simplePointers***.
  - Such diagrams help you understand pointers and are critical for tracing-pointers-by-hand problems.

```
double change;
double *pChange;

change = 0.45;
pChange = &change;
*pChange = 0.62;
```



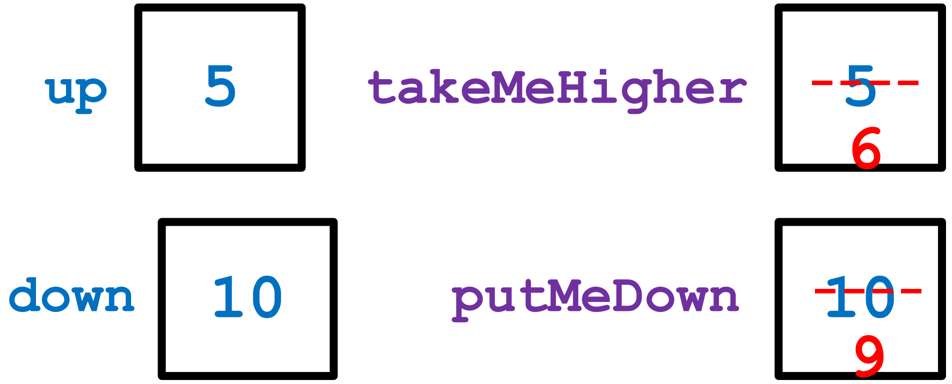
Q12

UpAndDown, WRONG version

- Do ***TODO 3*** in the program.
- Then do the quiz question (which asks you do draw a box-and-pointer diagram to explain how the following code executes):

```
int up = 5;
int down = 10;

upAndDownWrong(up, down);
```



```
void upAndDownWrong(int takeMeHigher, int putMeDown) {
    takeMeHigher = takeMeHigher + 1;
    putMeDown = putMeDown - 1;
}
```

Q13

Secret for making *upAndDownRight* – pass a *pointer* to the function

Goal of this slide: Show how a function can mutate a pointee in C

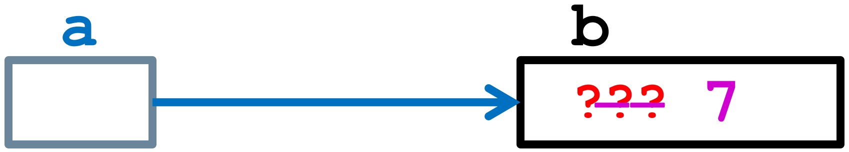
```
int b;
foo(&b);

void foo(int *a) {
     ...
     *a = 7;
}
```

Send the *address* of **b**

Receive an *address* via a *pointer*

Modify *value* at *address*, i.e., modify ***a's pointee***



Now **b** has the value **7** that was established in **foo**!

This is useful for:

- sending data back from a function via the parameters, and for
- passing large amounts of data to a function.

Thus pointers in C give us the same advantages as references-to-objects in Python.

UpAndDown, A version that works

- Do TODO 4, applying what you learned from the previous slide.
- When you are done, answer the quiz questions.

Q14-15

To read input from user in C, use `scanf()`

```
float x;
double y;
int z;

printf("Enter two real numbers and an integer:");
fflush(stdout);

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

printf("Average: %5.2f\n", (x + y + z) / 3.0);
```

In this use of `scanf`, user can enter the numbers separated by any whitespace (e.g. all on one line or on separate lines).

`fflush`: Pushes prompt string to user before asking for input.

Note `%lf` in `scanf` for double's.
Note `&`'s – see quiz question.
`scanf` is not resilient – if you misuse it, the compiler will generally not complain (your compiler is better than most) but the program will crash or simply give wrong results.

`scanf` has lots of options that are powerful but perhaps confusing – see pages 355-359 of Kochran if you need more structured input, and meanwhile stick to the above form, with ***spaces between the %'s.***

Q16

Summary: Why pointers are valuable

- If we pass pointers to a function:
  - ***The function can mutate the pointees.***
    - That is often convenient (although dangerous).
  - A `return` statement returns a *single* item.
    With pointer parameters, we can send back as many items as we have pointer parameters.
    - For example: `scanf` can send back multiple values
    - But: the single item in a `return` statement can be a structure instance, which "bundles" multiple pieces of data and returns them
      - While less convenient in C, perhaps, this is often the best approach in many languages.
  - ***The function can reference all data "just after" the pointer.***
    - *So it can reference many items without copying them – arrays (next time)*

Rest of today

- Work through the remaining TODO's, as numbered.
- Ask questions as needed!
  - Don't merely make the code "work". Make sure you understand the C notation and how to use it.
- ***Finish the exercises for homework***
  - ***Get help from the assistants in F-217, 7 to 11 p.m., as needed!***