

STUBS AND DRIVERS

CS133G
Beginning C++

PURPOSE FOR STUBS AND DRIVERS

- The reason for using stubs and drivers is to get your software working faster.
- You use stub functions to write a skeletal main. This is like the outline for your main program. Using stubs, you can compile your skeletal main and check it for syntax errors.
- You use drivers to test your functions. This way, you can write a function and test it before you have written the rest of the program.
- Once you have tested all your functions using the test drivers, you will be able to use those functions with your skeletal main. It will now be easier to flesh out your program and get it working since you won't need to spend as much time debugging.

- Stubs are just empty function definitions like this:

```
int getValidInteger(int min, int max)
{
    return 0;
}
```

- They are used so that you can write and compile your program's skeletal main without having to write the full definitions for all the functions.

STUBS

Functions with
an empty body

A SKELETAL MAIN

- A skeletal main is kind of like an outline of your program. Here's an example:

```
int main()
{
    showInstructions();
    const int NUM_SIDES = 6;    // number of sides on the die
    srand(time(NULL));
    int score = 0;
    do
    {
        int compRoll = rand() % NUM_SIDES + 1;
        cout << "Computer rolled " << compRoll << endl;
        score = playerRoll(NUM_SIDES, compRoll);
        cout << "It took you " << score << "rolls!" << endl;
    } while( getYesNo("Do you want to play again?") );

    return 0;
}
```

ESTABLISH AN ERROR FREE SKELETON

- You can compile the skeletal main to check for syntax errors by adding the function prototypes and stub functions.
- This is very useful since you will be able to catch any syntax errors early.
- It is extremely helpful to be able to debug your skeletal main so that when you have finished it, you know you are building on a syntactically correct framework.
- Note: while you can be sure there are no syntax errors, there could still be logic errors.
- There is an example of the main with prototypes and stub functions on the next slide:

```
int playerRoll(int numSides, int compRoll);
bool getYesNo(string question);
void showInstructions();

int main()
{
    showInstructions();
    const int NUM_SIDES = 6;    // number of sides on the die
    srand(time(NULL));
    int score = 0;
    do
    {
        int compRoll = rand() % NUM_SIDES + 1;
        cout << "Computer rolled " << compRoll << endl;
        score = playerRoll(NUM_SIDES, compRoll);
        cout << "It took you " << score << "rolls!" << endl;
    } while( getYesNo("Do you want to play again?" ) );
    return 0;
}

// Stubs
void showInstructions()
{
}

bool getYesNo(string question)
{
    return false;
}

int playerRoll(int numSides)
{
    return 3;
}
```

YOU'RE DONE WITH THE STUBS

- Once you finish your skeletal main and it compiles without errors, you are done with the stubs!
- Now you have a solid framework (or skeleton) on which to build your program. (Providing your logic was right.)

- A driver (or test driver) is just one or more lines of code in the program main that are used to test a function. For example:

```
cout << getYesNo("Do you want this to be  
true (y or n)?");
```

DRIVERS

A short block of code for testing a function.

TEST FUNCTIONS AS YOU WRITE THEM

- You will just use one test driver at a time to test one function as you are writing and debugging it.
- Comment out the skeletal code in your main before you write the first test driver. This way you can just run the driver by itself.
- If the function you are writing and testing doesn't return anything, you can test it by just calling it. Like this:
`showInstructions();`
- If the function returns a value, you can test it inside a cout statement like this:
`cout << playerRoll(6, 1) << endl;`
- Note that if your function has parameters, you will need to pass in some arguments when you call the function.

WRITE THE TEST DRIVER FIRST


- On the next slide, there is an example of a main with a test driver and a partially finished function.
- You can write and debug your function more quickly if you write the test driver first, then start writing your function.
- Testing as you write, speeds up your programming because you will be able to test your function after every two or three lines of code and catch syntax and logic errors early.
- Testing your functions before you use them in your program speeds up your programming because it is easier to test one function at a time rather than trying to test them along with all the rest of the functions and code in the main of your program.

```
// Prototypes
int playerRoll(int, int);
bool getYesNo(string);
void showInstructions();

int main()
{
    /*
    // Main program
    // code not shown to save space
    */

    /* Test drivers */
    cout << playerRoll(6, 1) << endl;
    // showInstructions();
    // cout << getYesNo("Enter y/n");

    return 0;
}
```



```
// Function definitions
```

```
int playerRoll(int numSides, int
compRoll)
{
    int roll = 0, score = 0;
    do
    {
        getYesNo("Do you want to roll
the die (y or n) ?");
        roll = rand() % numSides + 1;
        score++;
    } while (roll != compRoll);
    return score;
}

void showInstructions()
{
}

bool getYesNo(string question)
{
    return false;
}
```

NOW IT'S EASY TO FINISH THE PROGRAM

- Once you've finished writing and testing your function definitions you can comment out the test drivers and uncomment the skeletal main.
- Since you've tested all the functions and the skeletal main already, the program should compile without any syntax errors.
- Now all you have to do is test the logic and debug any problems in the program logic.